

# **Wireless Internet Platform for Interoperability 2.0.1**

**- Part 4. Java API -**

**September 2004**

**Korea Wireless Internet  
Standardization Forum**

1.	Overview of the Required Java API.....	2
2.	Foundation Class .....	4
2.1.	CLDC Class .....	4
2.2.	MSF Class .....	5
2.2.1.	Kernel .....	5
2.2.2.	Low-level I/O .....	15
3.	Profile .....	36
3.1.	MSP .....	36
3.1.1.	Graphics .....	36
3.1.2.	File .....	120
3.1.3.	Database .....	140
3.1.4.	High-Level User Interface .....	164
3.1.5.	Generic I/O .....	345
3.1.6.	Terminal Resource .....	348
3.1.7.	Address Book .....	360
3.1.8.	Media Handler .....	373
3.1.9.	SMS .....	418
3.1.10.	Location Information .....	422
3.1.11.	Control of Supplementary Devices .....	437
3.2.	MIDP .....	446
4.	Extended Unicode .....	447
4.1.	Extended Unicode Supporting Korean (EUC_KR) .....	447

# 1. Overview of the Required Java API

Java API consists of foundation classes and profiles.

A foundation class consists of the CLDC class that includes java.lang, java.io, and java.util as well as the kernel and low-level I/O methods of the MSF class.

A profile consists of MSP and MIDP classes. The major classes are shown below.

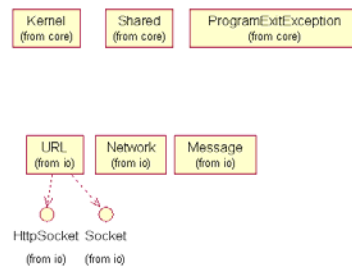


Figure 1-1. msf Diagram

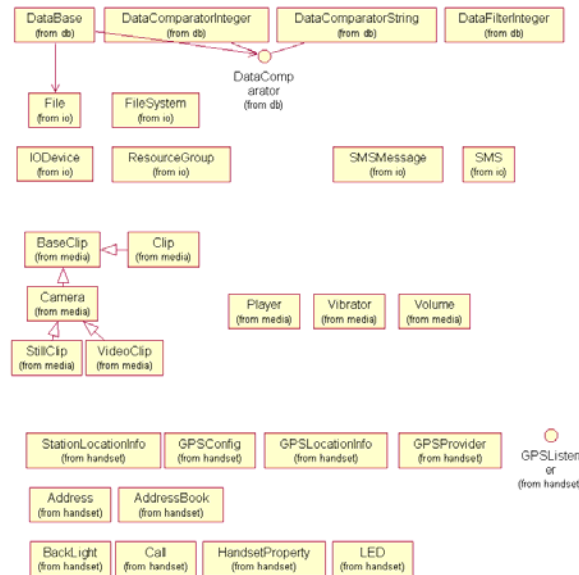


Figure 1-2. msp Diagram

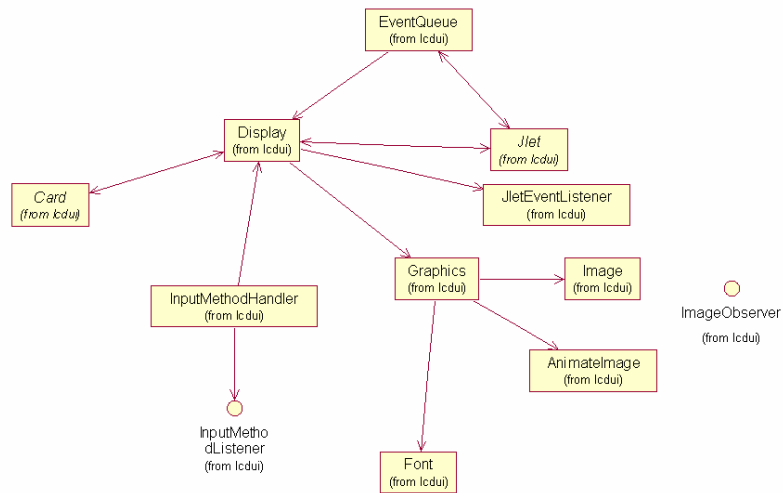


Figure 1-3. Icdi Diagram

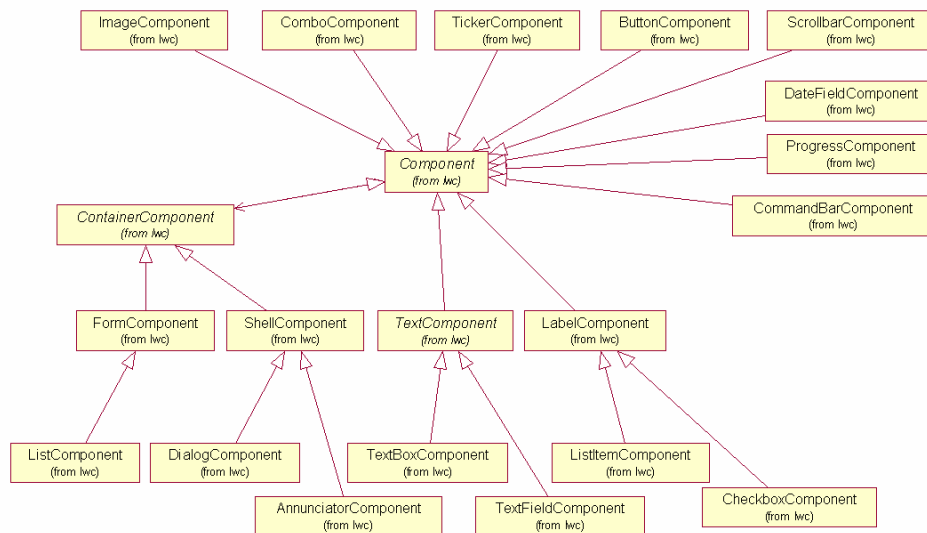


Figure 1-4. Iwc Diagram

## **2. Foundation Class**

### **2.1. CLDC Class**

For CLDC class specifications, either CLDC 1.1 (Specifications of Sun Microsystems) or identical methods should be provided.

Reference website: <http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html>

## 2.2. MSF Class

### 2.2.1. Kernel

#### Class Kernel

```

java.lang.Object
|
+--org.kwis.msf.core.Kernel

public class Kernel extends java.lang.Object

```

Kernel is a class providing system kernel methods. It imports program data, runs or exits a program, and loads a shared library.

#### Methods Inherited from Class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Field

- **PRGTYPE\_CAPP**

```
public static final int PRGTYPE_CAPP
```

C application

- **PRGTYPE\_CDLL**

```
public static final int PRGTYPE_CDLL
```

C dynamic loading library

- **PRGTYPE\_JAVAAPP**

```
public static final int PRGTYPE_JAVAAPP
```

Java application

- **PRGTYPE\_JAVADLL**

```
public static final int PRGTYPE_JAVADLL
```

Java dynamic loading library

- **PRGTYPE\_JAVASYDLL**

public static final int PRGTYPE\_JAVASYDLL

Java system library

- **DIR\_SYS\_READ\_REQ\_MASK**

public static final int DIR\_SYS\_READ\_REQ\_MASK

System directory read enabled

- **DIR\_SYS\_WRITE\_REQ\_MASK**

public static final int DIR\_SYS\_WRITE\_REQ\_MASK

System directory write enabled

- **DIR\_SHARED\_READ\_REQ\_MASK**

public static final int DIR\_SHARED\_READ\_REQ\_MASK

Shared directory read enabled

- **DIR\_SHARED\_WRITE\_REQ\_MASK**

public static final int DIR\_SHARED\_WRITE\_REQ\_MASK

Shared directory write enabled

- **NETWORK\_ACCESS\_REQ\_MASK**

public static final int NETWORK\_ACCESS\_REQ\_MASK

Network API enabled

- **SERIAL\_ACCESS\_REQ\_MASK**

public static final int SERIAL\_ACCESS\_REQ\_MASK

Serial API enabled

- **SYSTEM1\_ACCESS\_REQ\_MASK**

public static final int SYSTEM1\_ACCESS\_REQ\_MASK

APIs belonging to system group1 enabled (each communication service provider shall define the APIs to be included in system group1)

### ● **SYSTEM2\_ACCESS\_REQ\_MASK**

public static final int SYSTEM2\_ACCESS\_REQ\_MASK

APIs belonging to system group2 enabled (each communication service provider shall define the APIs to be included in system group2)

### ● **ACCESS\_ERROR**

public static final int ACCESS\_ERROR

This error occurs when a program to be executed has expired, or the user has no access right.

### ● **MEMORY\_ERROR**

public static final int MEMORY\_ERROR

This error occurs when a program cannot be executed due to insufficient memory.

### ● **ARGU\_ERROR**

public static final int ARGU\_ERROR

This error occurs when a program cannot be executed due to an invalid parameter transmitted.

## Detailed Description of the Generator

None

## Detailed Description of the Method

### ● **execute**

***public static int execute(String execName, String[] args)***

This method runs the program that is installed on a platform. When the program has expired, or the user has no access right, an error value will be returned. This is a non-blocking method. When the program is terminated, MV\_CHILDSTOP\_EVENT will be transmitted to the application program manager and to the parent of the program. The executed program and other programs are located in a separate memory space. Data can be exchanged through events and shared buffer only.

#### **Parameters**

execName	Name of the program to be executed; this can be obtained using the getExecNames() method
args	Parameter to be transmitted to the Main method()



**Return Value****Pass**

Program ID of the executed program

**Fail**

ACCESS_ERROR	When a program has expired, or the user has no access right
MEMORY_ERROR	In case of insufficient memory
ARGU_ERROR	When an invalid parameter is transmitted

**Reference Item**

getExecNames(String prgName, String version, String vendor)

- **load**

***public static int load(java.lang.String dllLibName, java.lang.String[] args)***

This method loads the Dynamic Linking Library (DLL) installed on the platform. If the DLL has expired, or access to the DLL is prohibited, an error value will be returned. This is a non-blocking method. If the program loads a library, the library will be located in the same memory as the program, and the library method can be called and used. If different programs load the same library, the library may be shared depending on the value (e.g., ADF) that is set during library registration or may be loaded separately. If all the programs that have loaded the library execute the MC\_knlUnload() method explicitly, or when all available programs end, the library is automatically terminated.

**Parameters**

dllLibName	Name of the program to be executed; this can be obtained using the getExecNames() method
args	Parameter to be transmitted to the Main method()

**Return Value****Pass**

Program ID of the loaded program

**Fail**

Negative value

**Reference Item**

getExecNames(String prgName, String version, String vendor)

- **load**

***public static int load(java.lang.String dllLibName)***

This method loads the Dynamic Linking Library (DLL) installed on the platform. If the DLL has expired, or access to the DLL is prohibited, an error value will be returned. This is a non-blocking method. If a program loads a library, the library will be located in the same memory as the program, and the library method can be called and used. If different programs load the same library, the library may be shared depending on the value (e.g., ADF) that is set during library registration or may be loaded separately. If all the programs that have loaded the library execute the MC\_knlUnload() method explicitly, or when all available programs end, the library is automatically terminated.

**Parameters**

dllLibName	Name of the program to be executed; this can be obtained using the getExecNames() method
------------	--

**Return Value**

**Pass**

ID of the loaded program

**Fail**

Negative value

**Reference Item**

getExecNames(String prgName, String version, String vendor)

- **getPrgID**

***public static int getPrgID()***

Gets the ID of the current program

**Return Value**

Program ID

- **getAMID**

***public static int getAMID()***

Gets the program ID of the application program manager

**Return Value**

Program ID

- **getParentPrgID**

***public static int getParentPrgID()***

Gets the ID of the parent program

**Return Value**

Program ID

- **getExecNames**

***public static String[] getExecNames(String prgName, String version,  
String vendor)***

This method returns an application identification name that matches prgName (program name), version, and vendor of the applications installed on the platform. A NULL parameter means that an identification name matches any of the prgName (program name), version, and vendor. For example, if the values for prgName, version, and vendor are all NULL, the names of all the programs installed on the platform will be returned. The names to be returned are a list of strings ending with null.

**Parameters**

prgName	Program name
version	Program version
vendor	Program vendor

**Return Value**

Program name string array

- **getPrgInfo**

***public static int[] getPrgInfo()***

This method gets information on currently running programs. A pair of program ID and type are included in a buf array to be returned. For example, if buf[0] is 1, and buf[1] is PRGTYPE\_ JAVADLL, the type of program whose ID is 1 will be java application DLL. Accordingly, the size of an array is twice as large as the number of programs.

**Return Value**

Integer array that includes prgID and type of a running program

- **stop**

***public static void stop(int prgID)***

This method forces a program to end. A DLL cannot be terminated forcibly. If all the APPs using a DLL end, the program will automatically end.

**Parameters**

prgID ID of a program to be terminated forcibly

- **letThrowExceptionWhenProgramExit**

***public static int letThrowExceptionWhenProgramExit()***

Once a program ends, ProgramExitException occurs in the thread that has called this method. This API is used for library development; when a program is terminated, this API enables a library to free the resources of the program being terminated.

**Return Value**

ID of the most recently terminated program (for a return value that is larger than 0)

- **getAccessLevel**

***public static int getAccessLevel()***

This method gets the access level of a program. Each bit of a return value indicates the types of APIs that can be accessed by the current program. The meaning of each bit is subject to the XXX\_REQ define statement defined above.

**Return Value**

Access level

- **getPrgName**

***public static java.lang.String getPrgName()***

Gets program names, written in the ADF file

**Return Value**

Program name

### Class ProgramExitException

```

java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.lang.RuntimeException
|
+--org.kwis.msf.core.ProgramExitException

```

All implemented interfaces: java.io.Serializable

public class **ProgramExitException** extends java.lang.RuntimeException

An exception that notifies the registered thread that a specific program has ended

#### Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

#### Detailed Description of the Generator

- **ProgramExitException**

***public ProgramExitException()***

Creates the ProgramExitException object

- **ProgramExitException**

***public ProgramExitException(java.lang.String s)***

Creates the ProgramExitException object with a message

**Class Shared**

```

java.lang.Object
|
+--org.kwis.msf.core.Shared

```

```

public class Shared extends java.lang.Object

```

This is a class that provides the memory shared among programs. The size of a shared buffer can be determined during creation, which is automatically freed when all programs using the shared buffer end.

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

- None

**Detailed Description of the Method**

- **createBuf**

***public static byte[] createBuf(String name, int size)***

This method creates shared buffers. A shared buffer is a byte array subject to multiple creation. The number of the shared buffer to be created depends on the platform. The created buffer is automatically deleted when all programs using the buffer are terminated. The type of information to be stored in the shared buffer is distinguished by storing the MIME type at the head of the buffer.

**Parameters**

name	Name of the shared buffer to be created
size	Size of the byte array buffer to be created

**Return Value**

Returns a created buf when the method succeeds; in the absence of a created buf, or if the method is no longer capable of creating a shared buffer, a null value is returned

- **getBuf**

***public static byte[] getBuf(String name)***

Gets the shared buffer

**Parameters**

name    Name of the shared buffer to be obtained

#### **Return Value**

Returns a shared buffer when the method succeeds; otherwise, if there is no shared buffer created, it returns a null value

#### ● **resizeBuf**

***public static byte[] resizeBuf(byte[] sharedBuf, int size)***

This method changes the size of a shared buffer. When the size becomes larger than the existing shared buffer, the content of the existing shared buffer is copied to the shared buffer whose size has been changed, and the remaining space is filled with 0. When the size becomes smaller than the existing shared buffer, however, only a portion of the content of the existing shared buffer can be copied to the shared buffer whose size has been changed. The shared buffer whose size has been changed is returned because new objects are assigned. Thus, programs possessing the existing shared buffer objects should get the shared buffers again.

#### **Parameters**

sharedBuf	Shared buffer reference whose size is to be changed
size	Size of the shared buffer to be changed

#### **Return Value**

Shared buffer whose size has been changed

#### ● **destroyBuf**

***public static void destroyBuf(byte[] sharedBuf)***

This method destroys the shared buffer created. When all programs sharing the shared buffer call destroyBuf(), the shared buffer is actually destroyed when destroyBuf() is called for the last time. Otherwise, it is automatically deleted when all programs using the shared buffer are destroyed.

#### **Parameters**

sharedBuf	Shared buffer reference to be destroyed
-----------	---

## 2.2.2. Low-level I/O

### Interface HttpSocket

public interface **HttpSocket**

Socket interface related to HTTP

#### Detailed Description of the Field

- **HEAD**

*public static final java.lang.String HEAD*

HTTP Head request method whose value is "HEAD"

- **GET**

*public static final java.lang.String GET*

HTTP Get request method whose value is "GET"

- **POST**

*public static final java.lang.String POST*

HTTP Post request method whose value is "POST"

- **TRACE**

*public static final java.lang.String TRACE*

HTTP Trace request method whose value is "TRACE"

- **PUT**

*public static final java.lang.String PUT*

HTTP Put request method whose value is "PUT"

- **DELETE**

*public static final java.lang.String DELETE*

HTTP Delete request method whose value is "DELETE"

- **OPTIONS**

*public static final java.lang.String OPTIONS*

HTTP Options request method whose value is "OPTIONS"

- **CONNECT**

*public static final java.lang.String CONNECT*

HTTP Connect request method whose value is "CONNECT"

- **HTTP\_OK**

*public static final int HTTP\_OK*



Server response code OK whose value is 200

- **HTTP\_CREATED**

***public static final int HTTP\_CREATED***

Server response code CREATED whose value is 201

- **HTTP\_ACCEPTED**

***public static final int HTTP\_ACCEPTED***

Server response code ACCEPTED whose value is 202

- **HTTP\_NON\_AUTHORITATIVE**

***public static final int HTTP\_NON\_AUTHORITATIVE***

Server response code NO AUTHORIZATION whose value is 203

- **HTTP\_NO\_CONTENT**

***public static final int HTTP\_NO\_CONTENT***

Server response code NO CONTENT whose value is 204

- **HTTP\_RESET\_CONTENT**

***public static final int HTTP\_RESET\_CONTENT***

Server response code RESET CONTENT whose value is 205

- **HTTP\_PARTIAL\_CONTENT**

***public static final int HTTP\_PARTIAL\_CONTENT***

Server response code PARTIAL CONTENT whose value is 206

- **HTTP\_MULTIPLE\_CHOICE**

***public static final int HTTP\_MULTIPLE\_CHOICE***

Server response code MULTIPLE CHOICE whose value is 300

- **HTTP\_MOVED\_PERMANENTLY**

***public static final int HTTP\_MOVED\_PERMANENTLY***

Server response code MOVED PERMANENTLY whose value is 301

- **HTTP\_MOVED\_TEMPORARILY**

***public static final int HTTP\_MOVED\_TEMPORARILY***

Server response code MOVED TEMPORARILY whose value is 302

- **HTTP\_SEE\_OTHER**

***public static final int HTTP\_SEE\_OTHER***

Server response code SEE OTHER whose value is 303

- **HTTP\_NOT\_MODIFIED**

***public static final int HTTP\_NOT\_MODIFIED***

Server response code NOT MODIFIED whose value is 304

- **HTTP\_USE\_PROXY**

***public static final int HTTP\_USE\_PROXY***

Server response code USE PROXY whose value is 305

- **HTTP\_BAD\_REQ**

***public static final int HTTP\_BAD\_REQ***

Server response code BAD REQUEST whose value is 400

- **HTTP\_UNAUTHORIZED**

***public static final int HTTP\_UNAUTHORIZED***

Server response code UNAUTHORIZED whose value is 401

- **HTTP\_PAYMENT\_REQUIRED**

***public static final int HTTP\_PAYMENT\_REQUIRED***

Server response code PAYMENT REQUIRED whose value is 402

- **HTTP\_FORBIDDEN**

***public static final int HTTP\_FORBIDDEN***

Server response code FORBIDDEN whose value is 403

- **HTTP\_NOT\_FOUND**

***public static final int HTTP\_NOT\_FOUND***

Server response code NOT FOUND whose value is 404

- **HTTP\_METHOD\_NOT\_ALLOWED**

***public static final int HTTP\_METHOD\_NOT\_ALLOWED***

Server response code METHOD NOT ALLOWED whose value is 405

- **HTTP\_NOT\_ACCEPTABLE**

***public static final int HTTP\_NOT\_ACCEPTABLE***

Server response code NOT ACCEPTABLE whose value is 406

- **HTTP\_PROXY\_AUTHENTICATION\_REQUIRED**

***public static final int HTTP\_PROXY\_AUTHENTICATION\_REQUIRED***

Server response code PROXY AUTHENTICATION REQUIRED whose value is 407

- **HTTP\_REQ\_TIMEOUT**

***public static final int HTTP\_REQ\_TIMEOUT***

Server response code REQUEST TIMEOUT whose value is 408

- **HTTP\_CONFLICT**

***public static final int HTTP\_CONFLICT***

Server response code CONFLICT whose value is 409

- **HTTP\_GONE**

***public static final int HTTP\_GONE***

Server response code GONE whose value is 410

- **HTTP\_LENGTH\_REQUIRED**

***public static final int HTTP\_LENGTH\_REQUIRED***

Server response code LENGTH REQUIRED whose value is 411

- **HTTP\_PRECONDITION\_FAILED**

***public static final int HTTP\_PRECONDITION\_FAILED***

Server response code PRECONDITION FAILED whose value is 412

- **HTTP\_ENTITY\_TOO\_LARGE**

***public static final int HTTP\_ENTITY\_TOO\_LARGE***

Server response code TOO LARGE whose value is 413

- **HTTP\_REQ\_TOO\_LONG**

***public static final int HTTP\_REQ\_TOO\_LONG***

Server response code REQUEST TOO LONG whose value is 414

- **HTTP\_UNSUPPORTED\_TYPE**

***public static final int HTTP\_UNSUPPORTED\_TYPE***

Server response code UNSUPPORTED TYPE whose value is 415

- **HTTP\_REQ\_RANGE**

***public static final int HTTP\_REQ\_RANGE***

Server response code REQUEST RANGE NOT SATISFIABLE whose value is 416

- **HTTP\_EXPECT\_FAIL**

***public static final int HTTP\_EXPECT\_FAIL***

Server response code EXPECTATION FAILED whose value is 417

- **HTTP\_SERVER\_ERR**

***public static final int HTTP\_SERVER\_ERR***

Server response code INTERNAL SERVER ERROR whose value is 500

- **HTTP\_NOT\_IMPL**

***public static final int HTTP\_NOT\_IMPL***

Server response code NOT IMPLEMENTED whose value is 501

- **HTTP\_BAD\_GATEWAY**

***public static final int HTTP\_BAD\_GATEWAY***

Server response code BAD GATEWAY whose value is 502

- **HTTP\_UNAVAILABLE**

***public static final int HTTP\_UNAVAILABLE***

Server response code UNAVAILABLE whose value is 503

- **HTTP\_GATEWAY\_TIMEOUT**

***public static final int HTTP\_GATEWAY\_TIMEOUT***

Server response code GATEWAY TIMEOUT whose value is 504

- **HTTP\_VERSION**

***public static final int HTTP\_VERSION***

Server response code HTTP VERSION whose value is 505

**Detailed Description of the Method**

- **close**

***public void close() throws java.io.IOException***

Closes the HTTP socket

**Throws**

Java.io.IOException                      Issued in case closing socket fails

- **getInputStream**

***public java.io.InputStream getInputStream() throws java.io.IOException***

Returns InputStream

**Return Value**

InputStream

**Throws**

Java.io.IOException                      Issued in case getting InputStream fails

- **getOutputStream**

***public java.io.OutputStream getOutputStream() throws java.io.IOException***

Returns OutputStream

**Return Value**

OutputStream

**Throws**

Java.io.IOException                      Issued in case getting OutputStream fails

- **getURL**

***public java.lang.String getURL()***

Returns the URL

**Return Value**

URL string

- **getProtocol**

***public java.lang.String getProtocol()***

Returns the protocol part of the URL

**Return Value**

Protocol string

- **getHost**

***public java.lang.String getHost()***

Returns the host part of the URL

**Return Value**

Host string

- **getFile**

***public java.lang.String getFile()***

Returns the file string of the URL

**Return Value**

File string

- **getRef**

***public java.lang.String getRef()***

Returns the anchor part of the URL

**Return Value**

Anchor string

- **getQuery**

***public java.lang.String getQuery()***

Returns the query part of the URL

**Return Value**

Query string

- **getPort**

***public int getPort()***

Returns the port part of the URL

**Return Value**

Port number with an integer type

- **getRequestMethod**

***public java.lang.String getRequestMethod()***

Returns the request method

**Return Value**

Request method string

- **setRequestMethod**

***public void setRequestMethod(java.lang.String method) throws  
java.io.IOException***

Sets the request method

**Parameters**

method          Request method string

**Throws**

Java.io.IOException          Issued in case setting request method fails

- **getRequestProperty**

***public java.lang.String getRequestProperty(java.lang.String key)***

Returns the relevant request property value to the parameter key

**Parameters**

key          Request header name

**Return Value**

Request property value

- **setRequestProperty**

***public void setRequestProperty(java.lang.String key, java.lang.String value)***

Sets the request property

**Parameters**

key          Request header name  
value          Request property value

- **getResponseCode**

***public int getResponseCode() Throws java.io.IOException***

Returns the server response code

**Return Value**

Integer response code

**Throws**

java.io.IOException    Issued when the response code is unknown

- **getResponseMessage**

***public java.lang.String getResponseMessage() throws java.io.IOException***

This method returns a response message from the server. When the response message from the server is HTTP/1.0 200 OK or HTTP/1.0 404 Not found, "OK" or "Not Found" is returned as a response message.

**Return Value**

Response message

**Throws**

java.io.IOException    Issued when the response message is unknown

- **getLength**

***public long getLength()***

Returns the length of the received content in byte

**Return Value**

Length of the received content

- **getType**

***public java.lang.String getType()***

Returns the type of the received content

**Return Value**

Content type string

- **getEncoding**

***public java.lang.String getEncoding()***

Returns the value corresponding to "content-encoding" among the response headers from the server

**Return Value**

Content encoding string

- **getExpiration**

***public long getExpiration()***



Returns the expiration date of the content

**Return Value**

Time from the GMT standard time of January 1, 1970 (millisecond)

- **getDate**

***public long getDate()***

Returns the date when the content is prepared

**Return Value**

Time from the GMT standard time of January 1, 1970 (millisecond)

- **getLastModified**

***public long getLastModified()***

Returns the time when the content is most recently modified

**Return Value**

Time from the GMT standard time of January 1, 1970 (millisecond)

- **getHeaderField**

***public java.lang.String getHeaderField(java.lang.String name)***

Returns the response header value from the server

**Parameters**

name    Response header

**Return Value**

Value corresponding to the response header

- **setProxy**

***public void setProxy(java.lang.String host, int port) throws  
java.io.IOException***

Specifies the HTTP proxy

**Parameters**

host    Proxy host

port    Proxy port

**Throws**

java.io.IOException    Issued when the proxy cannot be specified

- **isRelocatable**

***public boolean isRelocatable()***

Indicates when the server enables connection to another server as the location of the content is moved (this method is valid only when a response code is received from the server)

**Return Value**

Returns True when the response code from the server is 301, 302, or 303

- **relocation**

***public HttpSocket relocation() throws java.io.IOException***

The server enables connection to another server as the location of the content is moved. This method returns a new HTTP socket when connection is made to another server.

**Return Value**

HTTP socket with new connection

**Throws**

java.io.IOException    Issued when a new connection cannot be made

**Class Message**

java.lang.Object

|

+--org.kwis.msf.io.**Message**public class **Message** extends java.lang.Object

A class defining the message that can be transmitted to the socket

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**● **Message*****public Message(java.lang.String addr, byte[] data)***

Creates the message to be transmitted to the socket

**Parameters**

addr	Address for the message
data	Content of the message

● **Message*****public Message(java.lang.String addr, byte[] data, int off, int len)***

Creates the message to be transmitted to the socket

**Parameters**

addr	Message address
data	Message buffer
off	Offset of the message buffer
len	Length of the message buffer

**Detailed Description of the Method**● **getData*****public byte[] getData()***

Returns the message buffer

**Return Value**

Message buffer

● **getLength*****public int getLength()***

Returns the length of the message buffer

**Return Value**

Length of the message buffer

- **setLength**

***public int setLength(int val)***

This method sets the message length. When the aggregate of the length to be set and the return value of `getOffset()` exceed the buffer length returned by `getData()`, or the length is smaller than 0, the message length will not be set. Instead, -1 will be returned.

**Parameters**

val      Message length

**Return Value**

Set length

- **getOffset**

***public int getOffset()***

Returns the offset of the message buffer

**Return Value**

Offset of the message buffer

- **setOffset**

***public int setOffset(int val)***

This method sets the offset of the message buffer. When the aggregate of the offset to be set and the return value of `getLength()` exceed the buffer length returned by `getData()`, or the offset is smaller than 0 or longer than the buffer length returned by `getData()`, the offset of the message buffer will not be set. Instead, -1 will be returned.

**Parameters**

val      Offset

**Return Value**

Set offset

- **getAddress**

***public java.lang.String getAddress()***

Returns the address for the message

**Return Value**

Address for the message

- **getAddressInt**

***public int getAddressInt()***

Returns the integer address for the message

**Return Value**

Integer address for the message

- **getDate**

***public java.util.Date getDate()***

Returns the time when the message is transmitted

**Return Value****Pass**

Time when the message is transmitted

**Fail**

Returns a null value when the time of message transmission is unknown

- **setDate**

***public void setDate(java.util.Date date)***

Sets the time when the message is transmitted

**Parameters**

date                      Time when the message is transmitted

- **setAddressInt**

***public void setAddressInt(int addr)***

This method specifies the integer address for the message. The message address being transmitted to the UDP socket has the following format:

IP Address: Port number

Ex.) 111.111.111.111:80

**Parameters**

addr      Value for the integer address

**Class Network**

java.lang.Object

|

+---org.kwis.msf.io.**Network**

public class **Network** extends java.lang.Object

This is a collection of Internet access APIs used by an application program for TCP/IP Internet communication. An application program can use the TCP/IP communication socket only after Internet access is enabled through Internet access APIs. The TCP/IP communication socket is created using the URL.find() method.

**Detailed Description of the Method**

- **connect**

***public static int connect()***

Attempts TCP/IP Internet access

**Return Value**

Returns 0 when access is enabled; returns 1 when access is successful even without connection (-1 is returned if the attempt fails)

- **disconnect**

***public static void disconnect()***

This method terminates TCP/IP Internet access. Calling this method causes all I/O methods of the TCP/IP communication socket to be disabled.

### Class SchemeNotFoundException

```

java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.io.IOException
|
+--org.kwis.msf.io.SchemeNotFoundException

```

public class **SchemeNotFoundException** extends java.io.IOException

#### Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

#### Detailed Description of the Generator

- **SchemeNotFoundException**

*public SchemeNotFoundException()*

- **SchemeNotFoundException**

*public SchemeNotFoundException(java.lang.String s)*

## Interface Socket

public **interface** Socket

This is a class that enables data communication with the outside of the platform.

Socket is classified into stream socket and non-stream socket. A stream socket communicates with the outside of the platform using `java.io.InputStream` and `java.io.OutputStream`. On the other hand, a non-stream socket uses the message form. A notable example of stream socket is the one used in TCP communication, whereas a notable example of a message-type communication is a socket used in UDP communication.

## Detailed Description of the Method

- **getInputStream**

***public java.io.InputStream getInputStream() throws java.io.IOException***

Returns `InputStream` in a stream socket

**Return Value**

`InputStream`

**Throws**

<code>java.io.IOException</code>	Issued in case that it is a non-stream socket, or <code>InputStream</code> cannot be returned
----------------------------------	--

- **getOutputStream**

***public java.io.OutputStream getOutputStream() throws java.io.IOException***

Returns `OutputStream` in a stream socket

**Return Value**

Returns `OutputStream`

**Throws**

<code>java.io.IOException</code>	Issued in case that it is non-stream socket, or <code>OutputStream</code> cannot be returned
----------------------------------	---

- **isStream**

***public boolean isStream()***

Indicates whether the value is a stream socket

**Return Value**



True in case of a stream socket; otherwise, false is returned

- **getMessageCount**

***public int getMessageCount() throws java.io.IOException***

Indicates the number of messages that can be read from the socket (in case of a non-stream socket)

**Return Value**

Returns -1 when the number of messages that can be read in the future is unknown

**Throws**

java.io.IOException    Issued in case of a stream socket

- **getMessageMaxLength**

***public int getMessageMaxLength() throws java.io.IOException***

Indicates the maximum length of data that can be carried by a message (in case of a non-stream socket)

**Return Value**

Maximum length of message data

**Throws**

java.io.IOException    Issued in case of a stream socket

- **send**

***public void send(Message m) throws java.io.IOException***

Transmits a message in case of a non-stream socket

**Parameters**

m        Message where the data to be transmitted is stored

**Throws**

java.io.IOException        Issued in case of a stream socket, or a message cannot be transmitted

- **recv**

***public void recv(Message msg) throws java.io.IOException***

Receives a message in case of a non-stream socket

**Parameters**

m      Message where the received data is to be stored

**Return Value**

Received message

**Throws**

java.io.IOException      Issued in case of a stream socket, or a message  
cannot be transmitted

- **close**

***public void close() throws java.io.IOException***

This method closes the socket. In case of a stream socket, when InputStream or OutputStream is returned using getInputStream() or getOutputStream(), this method, together with close() of InputStream or OutputStream, should be called to enable the socket to be closed completely.

**Throws**

java.io.IOException      Issued in case of an error

- **accept**

***public Socket accept() throws java.io.IOException***

A socket supporting server methods; returns a socket that is connected with the client

**Return Value**

Newly connected socket

**Throws**

java.io.IOException      Issued if the socket does not support server  
methods, or an I/O error occurs

**Class URL**

```
java.lang.Object
```

```
|
```

```
+---org.kwis.msf.io.URL
```

```
public class URL extends java.lang.Object
```

This is a class that creates sockets to be used for data communication with the outside of the platform. All sockets are created by transmitting URL (see RFC 1738) strings to the URL.find() method. The character of the sockets created is distinguished by the scheme part of a URL string and divided into stream sockets and non-stream sockets. URL strings used in the sockets are defined in Table 2-2-1-1. The platform should create sockets for strings other than TCP server URLs (those beginning with serversocket://).

Table 2-2-1-1. URL Strings by Protocol

Socket	URL String	Example	Remarks
TCP	socket://<IP address>:<Port number>[/<Mode>/<Timeout>] serversocket://:<Port number>[/<Mode>/<Timeout>]	socket://111.111.111.111:80 socket://111.111.111.111:80/rw/3000	.Mode: r (Read-only), w (Write-only), rw (Read and Write) .Timeout: In millisecond as a unit; 0 means infinity .Mode and timeout apply when transmitting or receiving data. .When mode and timeout are omitted, the default values for mode and timeout are "rw" and "0," respectively. .In case of a server socket (those beginning with serversocket://), mode and timeout apply to the client socket for connection. . The option of supported server sockets is subject to platform implementation
UDP	datagram://:<Port number>[/<Mode>/<Timeout>]	datagram://:80 datagram://:80/rw/3000	.Bound to a local port .Mode and timeout have identical meaning with the TCP socket.
serial	comm://<Port number>:<Control string>	comm://0:baudrate=115200, parity=no, size=8, flow=no	.For control string, see the description of MH_serialOpen() in HAL API.
HTTP	HTTP URL	http://somehost:80	

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

None

**Detailed Description of the Method**● **find**

***public static Socket find(java.lang.String url) throws  
SchemeNotFoundException***

Creates sockets based on parameters url

**Throws**

SchemeNotFoundException Issued when sockets cannot be created

## 3. Profile

### 3.1. MSP

#### 3.1.1. Graphics

##### Interface ImageObserver

public interface **ImageObserver**

This is an interface that enables viewing image creation status. It is used to indicate the creation status for each image frame when images are created.

##### Detailed Description of the Field

- **FRAME\_END**

*public static final int FRAME\_END*

Constant indicating that an image frame succeeded in decoding

- **IMAGE\_END**

*public static final int IMAGE\_END*

Constant indicating that the entire image succeeded in decoding, or that the animation of the image has ended

- **NOT\_EXIST**

*public static final int NOT\_EXIST*

Constant indicating that the image cannot be read because the specified resource does not exist

- **DECODE\_ERROR**

*public static final int DECODE\_ERROR*

Constant indicating that decoding has failed because the data is invalid

- **OUT\_OF\_MEMORY**

*public static final int OUT\_OF\_MEMORY*

**Detailed Description of the Method**

- **notify**

***public void notify(Image img, int status)***

This method indicates that one image frame has been completed . It also notifies the result of decoding by status. The status value is FRAME\_END if a frame is completed, and IMAGE\_END if decoding of the entire image is completed. In case the decoding result is an error, the status is either NOT\_EXIST or DECODE\_ERROR OUT\_OF\_MEMORY.

**Reference Item**

Image.loadImage (java.lang.String, org.kwis.msp.lcdui.ImageObserver),  
Image.play (org.kwis.msp.lcdui.ImageObserver)

**Interface InputMethodListener**

```
public interface InputMethodListener
```

This is an interface used to detect the characters handled and the input status, i.e., insert, delete, and correct, of user key inputs in InputMethodHandler. The class that inherited and implemented InputMethodListener shall register it in InputMethodHandler.setInputMethodListener (org.kwis.msp.lcdui.InputMethodListener) of inputMethodHandler. In InputMethodHandler, the factors passed on to notifyTextChanged of InputMethodListener shall be the characters handled, the number of characters to be handled, and the handling status.

**Detailed Description of the Method**

- **notifyTextChanged**

***public void notifyTextChanged(char[] chText, int len, int pMode)***

Receives and handles the character objects transmitted by InputMethod

**Parameters**

chText	Input character	
len	Number of characters to be handled	
pMode	Handling status	Insert (-1) / replace (0) / delete (1)

**Interface JletEventListener**

public interface **JletEventListener**

This is an interface that processes application program (Jlet) events posted by postEvent. Basically, all events other than those posted by the input device should be processed by this Listener.

**Detailed Description of the Method**

- **notifyEvent**

***public void notifyEvent(int type, int param1, int param2)***

Called in case of application program events

**Parameters**

type	Event type
param1	Event parameter
param2	Event parameter

**Reference Item**

Display.addJletEventListener (org.kwis.msp.lcdui.JletEventListener)



**Class Card**

```
java.lang.Object
|
+--org.kwis.msp.lcdui.Card
```

Directly known subclasses: ProxyCard

```
public abstract class Card extends Object
```

This is a unit class that can be shown on the screen.

This class serves as a unit that can be shown on the screen, whereas a display consists of a stack of cards. Various cards stacked are shown on one display. One card cannot be put on several displays.

A card has its position and size on a display. The position or size of a card can be changed using the move or resize methods.

Using the repaint method, the paint method can be called again by an event handling thread for a part of the card; thus enabling the content to be shown on the display.

The card can receive user inputs. keyNotify and pointerNotify are some of the methods called by the user, and all events are first transmitted to an upper card in a stack. When the transmitted events are handled on the card, the method called above will return True, and the lower cards cannot receive any event. If False is returned, however, events can be transmitted to a lower card. Similarly, the lower cards that received events will return either True or False. The process continues up to the lowest card in the stack.

Keys inputted on a card basically include ITU-Keys 0 to 9 plus # and \*. These keys are required keys on any handset. Other keys can be distinguished using game keys. EventQueue.UP, EventQueue.DOWN, EventQueue.LEFT, EventQueue.RIGHT, and EventQueue.FIRE are some of the supported keys. EventQueue.SOFT1 and EventQueue.SOFT2 are also supported keys, although certain handsets do not support these keys. Compared to game keys, cross conversion is possible between key code and game key using Display.getGameAction(int) and Display.getKeyCode(int) methods.

When cards are shown or hidden using pushCard or popCard, the showNotify method is called.

For the coordinate system, the starting point is located on the upper left corner of a screen. When scrolling down from this point, the value on the Y axis increases. On the other hand, the value on the X axis increases when scrolling to the right from this point.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Detailed Description of the Field

- **x**

#### ***protected int x***

X coordinate of the card on the display

- **y**

#### ***protected int y***

Y coordinate of the card on the display

- **w**

#### ***protected int w***

Width of the card on the display

- **h**

#### ***protected int h***

Height of the card on the display

- **bTrans**

#### ***protected boolean bTrans***

This option specifies whether or not the card is transparent. Through this option, alpha blending can be used, or diverse forms of the card can be drawn. This is because the card is drawn after an immediately lower component of the present card finishes drawing.

### Detailed Description of the Generator

- **Card**

#### ***public Card()***

This method creates a card that is as large as the display. The size of the display is the same one returned using the Display.getDefaultDisplay method.

- **Card**

#### ***public Card(boolean bTrans)***

This method creates a card as large as the display. Transparency is determined based on bTrans. The size of the display is the same one returned using the Display.getDefaultDisplay method.

#### **Parameters**

bTrans                      Whether or not the card is transparent

- **Card**

***public Card(Display d)***

This method creates a card as large as the display. The card can be used only in a specified display.

**Parameters**

d        Display where a card is to be created

**Throws**

NullPointerException        Issued when d is null

**● Card*****public Card(int x, int y, int w, int h)***

This method creates a card using a specified size and position. A basic display is obtained using the Display.getDefaultDisplay method, and a card is created for the display. The size and position of the card are determined by the value of the argument transmitted.

**Parameters**

x        X coordinate of the card on the display  
y        Y coordinate of the card on the display  
w        Width of the card  
h        Height of the card

**Throws**

IllegalArgumentException        Issued when w or h is 0 or less

**● Card*****public Card(Display d, int x, int y, int w, int h)***

This method creates a card using the specified size and position for a specified display. The size and position of the card are determined by the value of the argument transmitted.

**Parameters**

d        Display where a card is to be created  
x        X coordinate of the card on the display

y      Y coordinate of the card on the display  
w      Width of the card  
h      Height of the card

**Throws**

IllegalArgumentException      Issued when w or h is 0 or less

- **Card**

***public Card(Display d, int x, int y, int w, int h, boolean bTrans)***

This method creates a card using the specified size and position for a specified display. The size and position of the card are determined by the value of the argument transmitted.

**Parameters**

d              Display where a card is to be created  
x              X coordinate of the card on the display  
y              Y coordinate of the card on the display  
w              Width of the card  
h              Height of the card  
bTrans        Whether or not the card is transparent

**Throws**

IllegalArgumentException      Issued when w or h is 0 or less

**Detailed Description of the Method**

- **move**

***public void move(int x, int y)***

Changes the position of a card on the display using specified values for x and y

**Parameters**

x              X coordinate of the card on the display  
y              Y coordinate of the card on the display

- **resize**

***public void resize(int w, int h)***

This method changes the size of a card. A w or h of 0 or less results in the IllegalArgumentException error.

**Parameters**

w      Width of the card  
h      Height of the card

**Throws**

IllegalArgumentException      Issued when w or h is 0 or less

- **getWidth**

***public int getWidth()***

Gets the width of a card on the display

**Return Value**

Width of the card

- **getHeight**

***public int getHeight()***

Gets the height of a card on the display

**Return Value**

Height of the card

- **getX**

***public int getX()***

Gets the x axis position of a card

**Return Value**

X coordinate of the card

- **getY**

***public int getY()***

Gets the y axis position of a card

**Return Value**

Y coordinate of the card

- **showNotify**

***protected void showNotify(boolean bShow)***

This method is called immediately before the card is shown on the display, with the bShow parameter set as True. On the other hand, this method is called with the bShow parameter set as False when the card is deleted from the display. In the inheriting class, it is necessary to put the method that registers or deletes an animation or timer in this method.

**Parameters**

bShow Whether or not the card is shown

- **keyNotify**

***protected boolean keyNotify(int type, int key)***

This method is called when the user key input is generated. When the user presses or releases a key, this method of the component with a focus is called.

When pressing or releasing a key, param1 becomes the value for the key code, and the internal sub-event type value such as KEY\_PRESSED or KEY\_RELEASED is passed on to the type.

In this method, when False is passed on as the return value, the event is transmitted to a lower card. Otherwise, when True is passed on, the event is no longer transmitted to the lower card.

**Parameters**

type KEY\_PRESSED, KEY\_RELEASED, KEY\_TYPED, or KEY\_REPEATED

key keyCode value: For more information on keyCode, see EventQueue

**Return Value**

True when transmitting the event to a lower card; otherwise, False is returned

- **pointerNotify**

***protected boolean pointerNotify(int type, int x, int y)***

This method is called when an input of the user pointing device is generated, i.e., when the user presses or releases a key, or when there is an input from the pointing device.

The type will be POINT\_PRESSED, POINT\_RELEASED, or POINT\_DRAGGED. The values of the x and y axes for the pointing device make up the coordinate system values on the card.

When this method passes on False as the return value, the event will be transmitted to a lower card. Otherwise, when True is passed on, the event will no longer be transmitted to the lower card.

In case of basic ITU keys, the corresponding ASCII code values will be used for KeyCode values; otherwise, negative values will be used. In case of control keys,

Display.getAction(int) will be used to determine whether they are corresponding keys.

**Parameters**

type	POINT_PRESSED, POINT_RELEASED, or POINT_DRAGGED
x	x coordinate of x Card in the display
y	y coordinate of y Card in the display

**Return Value**

True when transmitting the event to a lower card; otherwise, False is returned

- **paint**

***protected abstract void paint(Graphics g)***

This method paints the content of a card. Application programs require the implementation of this method. The g passed on as an argument is in the clipping to fit the card. Care should be taken when changing the clipping area using translate or setClip because it is designed to paint more than what is specified by the card. The content to be painted uses graphics objects.

**Parameters**

g	Graphics used for painting
---	----------------------------

- **repaint**

***public void repaint(int x, int y, int w, int h)***

This method repaints a specified area, i.e., the content of a rectangle beginning with x and y and up to width w and height h. Nonetheless, this method does not call the paint method directly; instead, the paint method calls it from the event handling thread after a specified time.

The graphics object passed on when calling the paint method is passed on again to the area to be painted. The clipping area can be larger than the area repainted, since it includes the area to be painted before this method is called.

The area to be repainted cannot exceed the area of the card.

**Parameters**

x	X coordinate indicating the specific area on the card
y	Y coordinate indicating the specific area on the card
w	Width of the specific area
h	Height of the specific area

- **repaint**

***public void repaint()***

This method repaints the entire area of the card. It has the same effect as calling `repaint(0, 0, getWidth() and getHeight())`.

- **serviceRepaints**

***public void serviceRepaints()***

This method repaints the repaint area forcibly and outputs it on the display. The paint method is directly called within this method.

- **isShown**

***public boolean isShown()***

This method indicates whether or not the card is shown on the display. The card should be registered on the display using the `pushCard` method for outputting on the display to enable this method to return `True`.

**Return Value**

Whether or not the card is shown

- **getDisplay**

***public Display getDisplay()***

Returns the display of the card

**Return Value**

Display of the card



**Class Display**

```
java.lang.Object
|
+--org.kwis.msp.lcd. Display
```

```
public class Display extends Object
```

This is a class with methods and information related to outputting on the display. Basically, it secures a display to output something on the display, creates a card, and calls the pushCard method to register the card on the display. The content painted using the paint method of the card is then outputted on the display.

One LCD responds to multiple displays. An application program can have multiple displays to enable the use of two displays on Jlet; thus supporting dual LCD models of the handset. To get displays, the getDisplay(java.lang.String) method is used.

One display consists of several cards. The card is painted one by one starting from the bottom-most card until the top-most card in a stack is finally painted. Dialog boxes are handled in a similar manner using this mechanism.

Inputting is carried out in the opposite direction. An input is transmitted to the top-most card, which indicates whether or not it handled the event. Once processed, the event is no longer transmitted to a lower card, otherwise it is. The event that is not handled by the top-most card is transmitted to a lower card to verify the handling status all the way down to the bottom-most card. Certain windows created by InputMethodHandler and other dialog boxes are actually handled in this manner.

Note that user input events (key event, pointer event, etc.) do not occur in displays that correspond to dual LCDs.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Method**

- **getDefaultDisplay**

```
public static Display getDefaultDisplay()
```

Gets the default display

**Return Value**

Default display

- **getD**
- **isplay**

***public static final Display getDisplay(String str)***

This method gets the display corresponding to a string. In case of a null string, this method gets the default display; in case a "dual" is passed on for the string depending on the phone, a display corresponding to dual LCD may be obtained. In the absence of a corresponding display, a null value is returned.

**Parameters**

str      String that indicates the display

**Return Value**

Display object

- **pushCard**

***public final void pushCard(Card c)***

This method causes a card to be shown on the display as well as a specified method to be called upon receipt of the user's input. It also causes the card to be positioned on top of the display for painting after all other cards are painted.

When the same card is already shown on the display, this method does not play any role. In case of a null c value, it does not carry out any operation.

When the card is shown on the display, the showNotify method is called prior to painting. Since the repaint method is then called, the paint method is called and shown on the display after a certain period of time.

Thereafter, isShown() always returns True.

When the application program calls pushCard() after executing another application program, the card will not be shown on top of the display; instead, it will be located immediately below such application program.

**Parameters**

c      Card to be shown on top of the display

**Throws**

NullPointerException      Issued when c is null

IllegalArgumentException      Issued in case of a different c display from the current display

- **popCard**

***public final Card popCard()***

This method takes out a card from the display to bring it out. In the absence of a card, a null value is returned. Only the card created by Jlet currently being implemented is removed.

When there is a card on the display, the showNotify() method is called before it can be removed from the display.

After that, False is returned when the isShown() method is called for the card returned by popCard().

**Return Value**

Removed card

- **removeCard**

***public final boolean removeCard(Card c)***

This method removes a specific card from the display to bring it out. This method is the same as popCard, although this method can specify a card. In case of a null c value, False is returned.

**Parameters**

c      Card to be removed

**Return Value**

Whether or not a card is normally removed from the stack

- **removeAllCards**

***public void removeAllCards()***

This method removes all cards created by Jlet currently being implemented from the display. When showNotify() for each card is called, the card is no longer shown on the display.

- **countCard**

***public final int countCard()***

Brings the number of cards registered by Jlet on the display

**Return Value**

Number of cards in the stack

- **callSerially**

***public final void callSerially(Runnable r)***

This is a method run for specific runnables when event processing is completed. In case of an event that calls a "Runnable" at the end of the event queue, the method is immediately terminated. When the event-processing thread processes this event, it runs the runnable class. Based on this structure, `r.run()` should have the shortest run time depending on the circumstances. In case of an infinite loop or a work that requires a considerable amount of time, however, event-related work cannot be processed. Consequently, the program cannot process user inputs properly.

**Parameters**

`r` Runnable object to be implemented

**Throws**

`NullPointerException` Issued when `r` is null

- **`callSerially`**

***public final void callSerially(Runnable r, int timeout)***

This is a method run for specific runnables when event handling is completed. In case of an event that calls a "Runnable" at the end of the event queue, the method is immediately terminated. When the event-processing thread processes this event, it runs the runnable class. Based on this structure, `r.run()` should have the shortest run time depending on the circumstances. In case of an infinite loop or a work that requires a considerable amount of time, however, event-related work cannot be processed. Consequently, the program cannot process user inputs properly. After a certain period of time, this method allows the calling of run of `r`. A timeout that is smaller than 0 will be regarded as 0.

**Parameters**

`r` Runnable object to be implemented

`timeout` Time when the runnable is called (millisecond)

**Throws**

`NullPointerException` Issued when `r` is null

- **`getDockedCard`**

***public Card getDockedCard()***

Returns a docked card

- **`setDockedCard`**

***public void setDockedCard(Card cd, int where)***

This method docks a card to a specific part of the display. This card is not covered by other cards. Once the card is set, the size of the display shrinks according to the remaining area.

Basically, the x and y values of the card are ignored. Where the value is TOP/BOTTOM, the width value is ignored. Where the value is LEFT/RIGHT, the height value is ignored. These values are based on the width/height value of the display.

A card registered as a push card should not exist. Only one Dockedcard can be set.

#### Parameters

cd	Card to be docked
where	Graphics on TOP, BOTTOM, LEFT, or RIGHT

#### Throws

NullPointerException	Issued when cd is null
IllegalArgumentException	Issued in case of a different cd display from the current display or incorrect value
IllegalStateException	Issued when more than one card is registered as pushCard or setDockedCard on display

#### ● isColor

***public final boolean isColor()***

Indicates whether or not the display supports colors

#### Return Value

Whether or not colors are supported

#### ● numColors

***public final int numColors()***

Indicates the number of colors available on the display

#### Return Value

Number of colors

- **hasPointerEvents**

***public final boolean hasPointerEvents()***

Indicates the presence or absence of a pointer device-related event in the system

**Return Value**

Whether or not there is a pointer event

- **hasPointerMotionEvents**

***public final boolean hasPointerMotionEvents()***

Indicates the presence or absence of a pointer motion device event

**Return Value**

Whether or not there is a pointer motion event

- **hasRepeatEvents**

***public final boolean hasRepeatEvents()***

Indicates whether or not key repeat events can occur

**Return Value**

Whether or not there is a key repeat event

- **getWidth**

***public final int getWidth()***

Returns the width of the display in pixel

**Return Value**

Width of the display

- **getHeight**

***public final int getHeight()***

Returns the height of the display in pixel

**Return Value**

Height of the display

- **isDoubleBuffered**

***public boolean isDoubleBuffered()***

This method indicates whether or not the display is double buffer. Most of the handsets support double buffering. Specifically, when painting in the graphics of the display, the content does not appear immediately on the display; instead, it appears only when a specific method (flush) is called.

**Return Value**

Whether or not the display is double buffer

- **getKeyCode**

***public static int getKeyCode(int gameKey)***

This method gets the KeyCode value corresponding to a gameKey. The gameKey may be EventQueue.UP, EventQueue.DOWN, EventQueue.LEFT, EventQueue.RIGHT, EventQueue.FIRE, EventQueue.GAME\_A, EventQueue.GAME\_B, EventQueue.GAME\_C, or EventQueue.GAME\_D. The value to be returned is the actual KeyCode value.

In case of an inappropriate gameKey, 0 is returned.

**Return Value**

Corresponding KeyCode value

**Reference Item**

EventQueue

- **getKeyName**

***public static String getKeyName(int key)***

This method receives the string of keyName corresponding to keyCode. For keyCode, refer to EventQueue. In case of an inappropriate key, a null value is returned. The returned string is in capital letters, e.g., "UP" and "DOWN." The string to be returned varies depending on the system.

**Return Value**

String of a corresponding keyName

**Reference Item**

EventQueue

- **getGameAction**

***public static int getGameAction(int key)***

This method gets the gameKey corresponding to a specified keyCode. System keyCodes are passed on, and they include ITU-Keys 0 to 9 plus \* and # as well as other control keys. Since control keys differ depending on the system, they should be

determined using the `getGameAction` method. For `keyCode`, refer to `EventQueue`.  
In case of an inappropriate key, 0 is returned.



**Return Value**

game Key

**Reference Item**

EventQueue

- **getBitsPerPixel**

***public int getBitsPerPixel()***

Returns bits constituting one pixel of the display

**Return Value**

Number of bits per pixel

- **flush**

***public void flush()***

The content of the internal buffer should be printed on the display. This method should be called when printing a painting carried out in specific graphics on the display. The serviceRepaints() method includes the internal flush() method and prints the content on the display when the isDoubleBuffer method returns True. Otherwise, it does not play any role.

- **addJletEventListener**

***public void addJletEventListener(JletEventListener qel)***

This method registers Listener, which receives JletEvent. All events other than key, repaint, and point register the Listener, which is called when the event occurs.

When re-registering a previously registered Listener, this method does not play any role.

**Parameters**

qel      Event Listener

- **removeJletEventListener**

***public void removeJletEventListener(JletEventListener qel)***

Deletes the Listener receiving JletEvent

**Parameters**

qel      Event Listener to be removed

- **grabKey**

***public void grabKey(int key, JletEventListener qel)***

A specific key is grabbed by the application program that calls this method. In case of an event for a specific key, the event for the key is passed on to the application program calling this method.

Only one grab is allowed for the same key. When the key is grabbed more than once, only the first grab will be considered.

Since this method grabs relevant keys regardless of application, it is recommended that the use of this method be limited to cases where AccessLevel is the system level. This method does not work in a program with a system level lower than AccessLevel.

**Parameters**

key      Current key

**● ungrabKey*****public void ungrabKey(int key)***

The possession of an event per grabKey is restored to the previous state and no longer maintained.

It is recommended that the use of this method be limited to cases where AccessLevel is the system level. This method does not work in a program with a system level lower than AccessLevel.

**Parameters**

key      Key to be ungrabbed

**Class EventQueue**

```
java.lang.Object
```

```
|
```

```
+--org.kwis.msp.lcdui.EventQueue
```

```
public class EventQueue extends Object
```

This is a queue class that manages events occurring in the system. There is one EventQueue object that manages events in one application program (Jlet). All events occurring in an application program are first stored in this object. The application program then brings the event one by one to process them as necessary; thus operating the application program. The thread bringing the events for appropriate processing exists within Jlet.

To bring an event or put in a new one, more integer arrays are used than what is specified by EVENT\_SIZE. The content to be put in this array varies depending on the event type.

The following is the content to be stored depending on the event type, as long as POINTER\_EVENT is an option in the absence of a pointing device like a touch screen:

Table 3-1-1. Storage content by event type

Event[0]	Event[1]	Event[2]	Event[3]
KEY_EVENT	KEY_PRESSED   KEY_RELEASED   KEY_REPEATED   KEY_TYPED	Key value (ASCII value or negative value in case of ITU keys)	0
POINTER_EVENT	POINTER_PRESSED   POINTER_RELEASED   POINTER_DRAGGED	X coordinate value of the pointer on the display	Y coordinate value of the pointer on the display
TIMER_EVENT	Timer value		

Within the Jlet application program being implemented, getNextEvent and dispatchEvent are executed infinitely. The following is the execution code:

```
int event[] = new int[EventQueue.EVENT_SIZE];

while(true){
    eq.getNextEvent(event);
    eq.dispatchEvent(event);
}
```

keyCode varies depending on the system. In the case of ITU-keys, however, the keys include 0 to 9 plus \* and # as well as other keys corresponding to the values of the ASCII code. Also included are the control keys such as power key or direction key, which are negative values. The values may vary depending on the platform. To handle different control keys based on platforms, the `getGameAction` and `getKeyCode` methods are used. In particular, to identify the type of key when a specific key is received, the `getGameAction` method is used. Depending on the system, number keys can also be used as control key. Currently, the `getKeyCode` method is the counterpart of `getGameAction`.

#### Methods inherited from class `java.lang.Object`

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

#### Detailed Description of the Field

- **EVENT\_SIZE**

***public static final int EVENT\_SIZE***

This is a unit size that stores one event, which is specified as 4. An integer array that is larger than 4 should be passed on when calling the `getNextEvent` method.

- **UP**

***public static final int UP***

Constant that specifies UP gameKey (1)

- **DOWN**

***public static final int DOWN***

Constant that specifies DOWN gameKey (6)

- **LEFT**

***public static final int LEFT***

Constant that specifies LEFT gameKey (2)

- **RIGHT**

***public static final int RIGHT***

Constant that specifies RIGHT gameKey (5)

- **FIRE**

***public static final int FIRE***

Constant that specifies FIRE gameKey (8)

- **GAME\_A**

***public static final int GAME\_A***

Constant that specifies GAME\_A gameKey (9)

- **GAME\_B**

***public static final int GAME\_B***

Constant that specifies GAME\_B gameKey (10)

- **GAME\_C**

***public static final int GAME\_C***

Constant that specifies GAME\_C gameKey (11)

- **GAME\_D**

***public static final int GAME\_D***

Constant that specifies GAME\_D gameKey (12)

- **KEY\_NUM0**

***public static final int KEY\_NUM0***

Constant that specifies the ITU-T '0' key (48 ('0'))

- **KEY\_NUM1**

***public static final int KEY\_NUM1***

Constant that specifies the ITU-T '1' key (49 ('1'))

- **KEY\_NUM2**

***public static final int KEY\_NUM2***

Constant that specifies the ITU-T '2' key (50 ('2'))

- **KEY\_NUM3**

***public static final int KEY\_NUM3***

Constant that specifies the ITU-T '3' key (51 ('3'))

- **KEY\_NUM4**

***public static final int KEY\_NUM4***

Constant that specifies the ITU-T '4' key (52 ('4'))

- **KEY\_NUM5**

***public static final int KEY\_NUM5***

Constant that specifies the ITU-T '5' key (53 ('5'))

- **KEY\_NUM6**

***public static final int KEY\_NUM6***

Constant that specifies the ITU-T '6' key (54 ('6'))

- **KEY\_NUM7**

***public static final int KEY\_NUM7***

Constant that specifies the ITU-T '7' key (55 ('7'))

- **KEY\_NUM8**

***public static final int KEY\_NUM8***

Constant that specifies the ITU-T '8' key (56 ('8'))

- **KEY\_NUM9**

***public static final int KEY\_NUM9***

Constant that specifies the ITU-T '9' key (57 ('9'))

- **KEY\_STAR**

***public static final int KEY\_STAR***

Constant that specifies the ITU-T '\*' key (42('\*'))

- **KEY\_POUND**

***public static final int KEY\_POUND***

Constant that specifies the ITU-T '#' key (35 ('#'))

- **KEY\_SEND**

***public static final int KEY\_SEND***

Call key (-10)

- **KEY\_END**

***public static final int KEY\_END***

End(Power) key (-11)

- **KEY\_CAMERA**

***public static final int KEY\_CAMERA***

Camera operating key (-19)

- **SOFT1**

***public static final int SOFT1***

Constant that specifies the SOFT1 gameKey (90)

- **SOFT2**

***public static final int SOFT2***

Constant that specifies the SOFT2 gameKey (91)

- **SOFT3**

***public static final int SOFT3***

Constant that specifies the SOFT3 gameKey (92)

- **SIDE\_UP**

***public static final int SIDE\_UP***

Constant that specifies the SIDE\_UP gameKey (96)

- **SIDE\_DOWN**

***public static final int SIDE\_DOWN***

Constant that specifies the SIDE\_DOWN gameKey (97)

- **SIDE\_SEL**

***public static final int SIDE\_SEL***

Constant that specifies the SIDE\_SEL gameKey (98)

- **CLEAR**

***public static final int CLEAR***

Constant that specifies the CLEAR gameKey (99)

- **KEY\_PRESSED**

***public static final int KEY\_PRESSED***

KEY\_PRESSED event-type constant (specified as 1)

- **KEY\_RELEASED**

***public static final int KEY\_RELEASED***

KEY\_RELEASED event-type constant (specified as 2)

- **KEY\_REPEATED**

***public static final int KEY\_REPEATED***

KEY\_REPEATED event-type constant (specified as 3)

- **KEY\_TYPED**

***public static final int KEY\_TYPED***

KEY\_TYPED event-type constant (specified as 4)

- **POINT\_PRESSED**

***public static final int POINT\_PRESSED***

POINT\_PRESSED event-type constant (specified as 1)

- **POINT\_RELEASED**

***public static final int POINT\_RELEASED***

POINT\_RELEASED event-type constant (specified as 2)

- **POINT\_DRAGGED**

***public static final int POINT\_DRAGGED***

POINT\_DRAGGED event-type constant (specified as 5)

- **KEY\_EVENT**

***public static final int KEY\_EVENT***

KEY\_EVENT constant (specified as 1)

- **POINTER\_EVENT**

***public static final int POINTER\_EVENT***

POINTER\_EVENT constant (specified as 2)

- **SMS\_EVENT**

***public static final int SMS\_EVENT***

SMS\_EVENT constant (specified as 4)

- **CALL\_EVENT**

***public static final int CALL\_EVENT***

Terminal CALL\_NOTIFY\_EVENT constant (specified as 7)

- **ANN\_EVENT**

***public static final int ANN\_EVENT***

Terminal INDICATOR\_NOTIFY\_EVENT constant (specified as 8)

- **REPAINT\_EVENT**

***public static final int REPAINT\_EVENT***

- **TIMER\_EVENT**

***public static final int TIMER\_EVENT***

TIMER\_EVENT constant (for internal use; specified as 42)



- **CALL\_SERIALLY\_EVENT**

***public static final int CALL\_SERIALLY\_EVENT***

Internal CALL\_SERIALLY\_EVENT constant (for internal use; specified as 43)

- **APP\_EVENT**

***public static final int APP\_EVENT***

APPLICATION PROGRAM\_Event constant (specified as 100)

- **CHILDSTART\_EVENT**

***public static final int CHILDSTART\_EVENT***

CHILD PROGRAM START\_EVENT constant (specified as 101)

- **CHILDSTOP\_EVENT**

***public static final int CHILDSTOP\_EVENT***

CHILD PROGRAM STOP\_EVENT constant (specified as 102)

- **ANNUNCIATOR\_CHANGE\_EVENT**

***public static final int ANNUNCIATOR\_CHANGE\_EVENT***

PARENT ANNUNCIATOR\_CHANGE\_EVENT constant (specified as 103)

- **USER\_EVENT**

***public static final int USER\_EVENT***

USER\_EVENT constant (specified as 0 x 5000)

- **APP\_STOP**

***public static final int APP\_STOP***

APPLICATION PROGRAM\_STOP\_EVENT; This is a sub-type of APP\_EVENT specified as 1

- **APP\_RESUME**

***public static final int APP\_RESUME***

APPLICATION PROGRAM\_RESUME\_EVENT; this is a sub-type of APP\_EVENT specified as 2

- **APP\_DESTROY**

***public static final int APP\_DESTROY***

APPLICATION PROGRAM END\_EVENT; this is a sub-type of APP\_EVENT specified as 3

- **APP\_ACTIVE**

***public static final int APP\_ACTIVE***

Changes the currently active program to itself (for internal use; specified as 4)

### Detailed Description of the Method

- **getNextEvent**

***public void getNextEvent(int[] event)***

This method gets the new event that occurred. The new event is copied to an integer array. The event type is first stored in the array, followed by various contents based on such event type.

The size of an event array should be larger than EVENT\_SIZE. Otherwise, ArrayIndexOutOfBoundsException is issued.

This method should be called within an event-processing thread. Otherwise, IllegalStateException is issued.

The event-processing thread is the one calling methods such as keyNotify of the Card and paint.

**Parameters**

event    Array where an event is to be stored (should be larger than EVENT\_SIZE)

**Throws**

ArrayIndexOutOfBoundsException	Issued when the size of an event array is smaller than EVENT_SIZE
IllegalThreadStateException	Issued when this method is called from a thread that is not an event-processing thread
NullPointerException	When the event is null

- **postEvent**

***public boolean postEvent(int[] event)***

This method posts a new event to be processed by the event-processing thread.

When the event is posted in an array, its content is copied. The size of the array should be larger than EVENT\_SIZE.

False is returned if the event queue is completely filled; otherwise, True is returned if it is properly stacked in the event queue.

**Parameters**

event    Event to be posted

**Return Value**

True if properly posted; otherwise, False is returned

**Throws**

ArrayIndexOutOfBoundsException    Issued when the size of the event array is smaller than EVENT\_SIZE

NullPointerException    Issued when the event is null

- **postEvent**

***public static void postEvent(int id, int[] event)***

An event is transmitted to a specific application program being executed with ID, which is indicated by an ID.

Given an ID of -1, parameters are transmitted to a currently active program. When the ID has other values, however, the event is ignored. The event being transmitted should not be a system event.

**Parameters**

id    ID of an application program being executed

event    Event

- **dispatchEvent**

***public void dispatchEvent(int[] event)***

This method handles and passes on an event to be processed by an event-processing thread existing within the system.

**Parameters**

event    Event to be processed

**Throws**

ArrayIndexOutOfBoundsException    Issued when the size of the event array is smaller than EVENT\_SIZE

**Class Font**

```

java.lang.Object
|
+--org.kwis.msp.lcdui.Font

public class Font extends Object

```

This is a font class representing characters that are used to print strings. Select the desired font using `getFont`. Basically, since the font selected by calling `getFont` is internally shared, there is no need to free resources.

Select the desired font using the `getFont` method, specify the font type using the `Graphics.setFont (org.kwis.msp.lcdui.Font)` method, and change the outer look of the characters on the display.

Depending on the system, a similar font that is not the one specified can be received using `getFont`.

To determine the length of a string on the display, use the `stringWidth` or `substringWidth` method.

The style of the font shall be any one of the following:

```

STYLE_PLAIN
STYLE_UNDERLINED
STYLE_BOLD
STYLE_BOLD|STYLE_UNDERLINED
STYLE_ITALIC
STYLE_ITALIC|STYLE_UNDERLINED
STYLE_BOLD|STYLE_ITALIC
STYLE_BOLD| STYLE_ITALIC | STYLE_UNDERLINED

```

**Methods inherited from class java.lang.Object**

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

**Detailed Description of the Field**

- **FACE\_MONOSPACE**

***public static int FACE\_MONOSPACE***

Monospace font face (value is set to 32; specifies a font with a fixed width)

- **FACE\_PROPORTIONAL**

***public static int FACE\_PROPORTIONAL***

Proportional font face (value is set to 64; specifies a font with variable width)

- **FACE\_SYSTEM**

***public static int FACE\_SYSTEM***

System font face (value is set to 0; this is the font used by the system, and it is either MONOSPACE or PROPORTIONAL)

- **SIZE\_LARGE**

***public static int SIZE\_LARGE***

Large font (value is set to 16; specifies a large font in the system)

- **SIZE\_MEDIUM**

***public static int SIZE\_MEDIUM***

Medium font (value is set to 0; specifies a medium font in the system)

- **SIZE\_SMALL**

***public static int SIZE\_SMALL***

Small font (value is set to 8; specifies a small font in the system)

- **STYLE\_BOLD**

***public static int STYLE\_BOLD***

Bold font style (can be used in combination with other font styles; value is set to 1)

- **STYLE\_ITALIC**

***public static int STYLE\_ITALIC***

Slanting font style (can be used in combination with other font styles; value is set to 2)

- **STYLE\_PLAIN**

***public static int STYLE\_PLAIN***

Plain font style (can be used in combination with other font styles; value is set to 0)

- **STYLE\_UNDERLINED**

***public static int STYLE\_UNDERLINED***

Underlined font style (can be used in combination with other font styles; value is set to 4)

- **style**

***protected int style***

Field where the font style is stored

- **size**

***protected int size***

Field where the font size is stored

- **face**

***protected int face***

Field where the font face is stored

**Detailed Description of the Method**

- **getFont**

***public static Font getFont(int face, int style, int size)***

This method gets a font that is most similar to the specified font. Depending on the system, this method can obtain a proper font.

**Parameters**

face	Font face: Value for FACE_MONOSPACE, FACE_PROPORTIONAL, or FACE_SYSTEM
style	Font style: A combination of STYLE_PLAIN, STYLE_ITALIC, STYLE_BOLD, and STYLE_UNDERLINED
size	Font size: SIZE_LARGE, SIZE_MEDIUM, or SIZE_SMALL

**Throws**

IllegalArgumentException	Issued when any of the face, style, or size has an invalid value
--------------------------	--

- **charsWidth**

***public int charsWidth(char[] ch, int offset, int length)***

This method passes on the width of a string on the display in pixel. The string to be calculated includes characters from the offset of the string array as indicated by the characters to the length of characters.

**Parameters**

ch      Character array  
offset   Position in the character array  
length   Number of characters

**Return Value**

Width of the string on the display in pixel

**Throws**

ArrayIndexOutOfBoundsException	Issued when the offset is smaller than 0, or the length of data is larger than 1, when the length is smaller than 0, or when offset + length is larger than the length of data
IllegalArgumentException	Issued when the length is 0
NullPointerException	Issued when ch is null

- **charWidth**

***public int charWidth(char ch)***

This method passes on the width of a character as represented by ch on the display in pixel. The width " "(0 x 20) is passed on in case of a character that is not supported by the system. During printing on the display, " " is printed.

**Parameters**

ch      Character

**Return Value**

Width of the character on the display in pixel

- **getBaselinePosition**

***public int getBaselinePosition()***

Returns the base line position (height) of the character

**Return Value**

Baseline of the character

- **getDefaultFont**

***public static Font getDefaultFont()***

Restores the default font of the system

**Return Value**

Default font of the system

- **getFace**

***public int getFace()***

This method returns the font face. Depending on the font, the value to be returned is FACE\_MONOSPACE, FACE\_PROPORTIONAL, or FACE\_SYSTEM.

**Return Value**

Font face

- **getHeight**

***public int getHeight()***

Gets the font height

**Return Value**

Font height

- **getSize**

***public int getSize()***

Gets the font size

**Return Value**

Font size



- **getStyle**

***public int getStyle()***

This method gets the font style. Depending on the font style, a value for STYLE\_BOLD, STYLE\_ITALIC, STYLE\_UNDERLINE, or STYLE\_PLAIN is returned.

**Return Value**

Font style

- **isBold**

***public boolean isBold()***

Indicates whether or not the font style is STYLE\_BOLD

**Return Value**

True when the font style is STYLE\_BOLD; otherwise, False is returned

- **isItalic**

***public boolean isItalic()***

Indicates whether or not the font style is STYLE\_ITALIC

**Return Value**

True when the font style is STYLE\_ITALIC; otherwise, False is returned

- **isPlain**

***public boolean isPlain()***

Indicates whether or not the font style is STYLE\_PLAIN

**Return Value**

True when the font style is STYLE\_PLAIN; otherwise, False is returned

- **isUnderlined**

***public boolean isUnderlined()***

Indicates whether or not the font style is STYLE\_UNDERLINED

**Return Value**

True when the font style is STYLE\_UNDERLINED; otherwise, False is returned

- **stringWidth**

***public int stringWidth(String str)***

Gets the width of the string specified by str on the display

**Parameters**

str      String whose width is to be calculated

**Return Value**

Width of the string

**Throws**

NullPointerException      Issued when str is null

- **substringWidth**

***public int substringWidth(String str, int offset, int len)***

Gets the width of a part of a string specified by str from offset to len

**Parameters**

str              String whose width is to be calculated

offset          Starting point of the string

len              Number of characters in the string

**Return Value**

Width of the string

**Throws**

StringIndexOutOfBoundsException      Issued when the values of offset and len  
are out of bounds for string

NullPointerException      Issued when str is null

## Class Graphics

```
java.lang.Object
|
+--org.kwis.msp.lcdui.Graphics
```

```
public class Graphics extends Object
```

This class provides simple methods that can draw geometric figures such as text box, image, line, rectangle, and arc. Rectangle and arc can be painted with a specific color. In particular, rectangle can have round corners.

### Coordinate System

In this coordinate system, the starting point (0, 0) is located on the upper left corner, with the y axis increasing downward and the x axis increasing to the right. All coordinates used in a graphic object are in a coordinate system with a starting point that can be changed using the translate method.

### Anchor

The anchor serves as a parameter that determines the position of an image or a font during printing. Specifically, the anchor determines which part of an object should be positioned in specified coordinates.

For font, LEFT, HCENTER, or RIGHT can be used as the horizontal anchor, and TOP, BASELINE, or BOTTOM can serve as the vertical anchor. In specifying the anchor, logical OR is used to represent said horizontal and vertical contents.

For image, VCENTER is used as the anchor instead of BASELINE. At least one horizontal anchor should be specified, although the vertical anchor constituting an OR to the horizontal anchor need not be specified.

### Stroke Style

Stroke style is defined as either DOTTED or SOLID. It applies to drawLine and drawArc except paint methods like fillRect.

### Other Drawing Modes

Aside from stroke style, there are other drawing modes supported by Graphics, such as a drawing model called XOR and another that specifies the degree of transparency. The setXORMode(boolean) method enables the XOR printing of the display content and the content of the current printing. In contrast, the setAlpha(int) method enables the mixed printing of the display content and the content of the current printing. Typically, g.setAlpha(255) prints the content on the display, unlike g.setAlpha(0). When printing in XOR

mode or Alpha mode, speed decreases. All graphic operations are affected by the clipping area. Outside of the clipping area, however, the content is not changed by operations. The clipping area cannot be larger than the size of the display or image used when the graphic object is created.

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Field

- **BASELINE**

***public static final int BASELINE***

Constant specifying the anchor position at the baseline of a string (64)

- **BOTTOM**

***public static final int BOTTOM***

Constant specifying the anchor position below a character or an image (32)

- **DOTTED**

***public static final int DOTTED***

Constant specifying the dot stroke style (1)

- **HCENTER**

***public static final int HCENTER***

Constant specifying the horizontal position of the anchor at the center of a character or an image (1)

- **LEFT**

***public static final int LEFT***

Constant specifying the horizontal position of the anchor to the left of a character or an image (4)

- **RIGHT**

***public static final int RIGHT***

Constant specifying the horizontal position of the anchor to the right of a character or an image (8)

- **SOLID**

***public static final int SOLID***

Constant specifying the solid stroke style (0)

- **TOP**

***public static final int TOP***

Constant specifying the vertical position of the anchor to the top of a character or an image (16)

- **VCENTER**

***public static final int VCENTER***

Constant specifying the vertical position of the anchor at the center of an image (2)

**Detailed Description of the Method**

- **clipRect**

***public void clipRect(int x, int y, int width, int height)***

This method specifies the common area between a clipping area and a specified rectangle as the clipping area.

The internal clipping rectangle is the largest rectangle that can be included in a rectangle starting with the current (x, y) coordinates of a graphic object with height and width as well as in an internal clipping rectangle.

**Parameters**

x	X coordinate in the graphics coordinate system of the rectangle to be intersected
y	Y coordinate in the graphics coordinate system of the rectangle to be intersected
width	Width of the rectangle to be intersected
height	Height of the rectangle to be intersected

**Throws**

IllegalArgumentException	Issued in case of negative width and height values
--------------------------	--

**Reference Item**

setClip(int, int, int, int)

- **drawChar**

***public void drawChar(char character, int x, int y, int anchor)***

Draws a character specified by character in a specified position using the font and the color currently utilized by a graphic object in the graphic coordinate system

**Parameters**

character	Character to be drawn
x	X coordinate of the anchor point
y	Y coordinate of the anchor point
anchor	Anchor position of the text; see Anchor

**Throws**

IllegalArgumentException	Issued when the anchor value is not valid
--------------------------	---

**Reference Item**

drawChars(char[], int, int, int, int, int)

- **drawChars**

***public void drawChars(char[] data, int offset, int length, int x, int y, int anchor)***

Draws a part of the string indicated by data in a specified position using the font and the color currently utilized by a graphic object

**Parameters**

data	String to be drawn
offset	Starting position of the string to be drawn
length	Number of strings
x	X coordinate of the anchor point
y	Y coordinate of the anchor point
anchor	Anchor position of the text; see Anchor

**Throws**

ArrayIndexOutOfBoundsException	Issued when the values of offset and length are out of bounds for the data array
IllegalArgumentException	Issued when the anchor value is not valid
NullPointerException	Issued when the data value is null

**Reference Item**

drawString(java.lang.String, int, int, int)

- **drawImage**

***public void drawImage(Image img, int x, int y, int anchor)***

Draws an image indicated by img in a specified position

**Parameters**

img	Image to be drawn
x	X coordinate of the anchor point
y	Y coordinate of the anchor point
anchor	Anchor position of the text; see Anchor

**Throws**

IllegalArgumentException	Issued when the anchor value is not valid
NullPointerException	Issued when img is null

**Reference Item**

Image

- **drawLine**

***public void drawLine(int x1, int y1, int x2, int y2)***

Draws a connecting line between two points in the coordinate system of the current graphics using the color and stroke style defined by the graphic object

**Parameters**

x1	X coordinate of the starting point of a line
y1	Y coordinate of the starting point of a line
x2	X coordinate of the end point of a line
y2	Y coordinate of the end point of a line

- **drawRect**

***public void drawRect(int x, int y, int width, int height)***

This method draws a rectangle using the color and stroke style specified by the current graphics.

The area drawn by this method has a width of (width + 1) and a height of (height + 1). Nothing is drawn when the width or height is smaller than 0.

Unlike fillRect, the interior is not painted during drawing.

**Parameters**

x	X coordinate to be used in drawing the rectangle
y	Y coordinate to be used in drawing the rectangle
width	Width of the rectangle
height	Height of the rectangle

**Reference Item**

```
fillRect(int, int, int, int)
```

- **drawRoundRect**

```
public void drawRoundRect(int x, int y, int width, int height, int arcWidth,  
                           int arcHeight)
```

This method draws a rectangle with round corners using the color and stroke style specified by the current graphics.

The area drawn by this method has a width of (width + 1) and a height of (height + 1). Nothing is drawn when the width or height is smaller than 0.

Unlike fillRoundRect, the interior is not painted during drawing. When arcWidth is larger than width/2, it becomes width/2. Similarly, when arcHeight is larger than height/2, it becomes height/2.

**Parameters**

x	X coordinate to be used in drawing the rectangle
y	Y coordinate to be used in drawing the rectangle
width	Width of the rectangle
height	Height of the rectangle
arcWidth	Horizontal radius of the arc to be drawn on four corners
arcHeight	Vertical radius of the arc to be drawn on four corners

**Reference Item**

```
fillRoundRect(int, int, int, int, int, int)
```

- **drawString**

```
public void drawString(String str, int x, int y, int anchor)
```

This method draws a string using the color and stroke style specified by the current graphics. Any character that cannot be drawn in the string will be treated as space.

**Parameters**

str	String to be drawn
x	X coordinate of the anchor point
y	Y coordinate of the anchor point
anchor	Anchor position of the text; see Anchor

**Throws**

IllegalArgumentException	Issued when the anchor value is not valid
NullPointerException	Issued when str is null

**Reference Item**



```
drawChars(char[], int, int, int, int, int), drawSubstring(String, int, int, int, int, int)
```

## ● drawSubstring

***public void drawSubstring(String str, int offset, int len, int x, int y, int anchor)***

This method draws a part of the string using the color and font specified by the current graphics. Any character that cannot be drawn in the string will be treated as space. .

### Parameters

str	String to be drawn
offset	Starting interface counting from 0 within the string to be drawn
len	Number of characters in a part of the string
x	X coordinate of the anchor point
y	Y coordinate of the anchor point
anchor	Anchor position of the text; see Anchor

### Throws

IllegalArgumentException	Issued when the anchor value is not valid
NullPointerException	Issued when str is null
StringIndexOutOfBoundsException	Issued when the values of offset and len cannot indicate the internal string of str

### Reference Item

```
drawString(String, int, int, int)
```

## ● drawArc

***public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)***

This method draws an arc using the color and stroke style specified by the current graphics.

This method draws an arc starting from startAngle up to arcAngle. The angle will be in the direction where the X coordinate increases and in the direction where Y coordinate does not increase from 0 (3 o'clock direction). Clockwise, it will be a positive value; counter-clockwise, it will be negative value.

The center of the arc shall be the center of a rectangle having a size specified by the starting point (x, y), height, and width. Similar to drawRect, this will occupy an area with width of (width + 1) and height of (height + 1). Nothing will be drawn when either the width or the height is smaller than 0.

### Parameters

x	X coordinate of the upper left corner of the arc
---	--

y	Y coordinate of the upper left corner of the arc
width	Width of the arc when it is drawn
height	Height of the arc when it is drawn
startAngle	Starting angle
arcAngle	Angle of the arc width from the starting angle

**Reference Item**

```
fillArc(int, int, int, int, int, int)
```

- **fillArc**

***public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)***

This method paints an arc using the color specified by the current graphics.

This method draws an arc as specified by startAngle and arcAngle. For the position, the same drawing method used in drawArc shall be employed.

Similar to the case of fillRect, the arc will occupy an area with width and height.

The painted area will have a pie shape. Nothing will be drawn when either the width or the height is 0 or less.

**Parameters**

x	X coordinate of the upper left corner of the arc
y	Y coordinate of the upper left corner of the arc
width	Width of the arc when it is drawn
height	Height of the arc when it is drawn
startAngle	Starting angle
arcAngle	Angle of the arc width from the starting angle

**Reference Item**

```
fillArc(int, int, int, int, int, int)
```

- **fillRect**

***public void fillRect(int x, int y, int width, int height)***

This method paints the rectangle using the color specified by the current graphics.

The area painted using this method will have an area with width and height. Nothing will be painted when either the width or height is 0 or less. The interior will be painted during drawing.

**Parameters**

x	X coordinate to be used in drawing the rectangle
y	Y coordinate to be used in drawing the rectangle
width	Width of the rectangle

height            Height of the rectangle

#### Reference Item

drawRect(int, int, int, int)

#### ● fillRoundRect

***public void fillRoundRect(int x, int y, int width, int height, int arcWidth,  
int arcHeight)***

This method paints a rectangle with round corners using the color specified by the current graphics.

The area painted using this method has a width of (width + 1) and a height of (height + 1). Nothing will be painted when the width or height is smaller than 0.

The interior is painted during drawing. When arcWidth is larger than width/2, it becomes width/2. Similarly, when arcHeight is larger than height/2, it becomes height/2.

#### Parameters

x	X coordinate to be used in drawing the rectangle
y	Y coordinate to be used in drawing the rectangle
width	Width of the rectangle
height	Height of the rectangle
arcWidth	Horizontal radius of the arc to be drawn on four corners
arcHeight	Vertical radius of the arc to be drawn on four corners

#### Reference Item

drawRoundRect(int, int, int, int, int, int)

#### ● fillPolygon

***public void fillPolygon(int[] x, int[] y)***

Draws a polygon using specified (x,y) points and fills the interior with the color specified by the current graphics

#### Parameters

x	X array of the polygon to be drawn
y	Y array of the polygon to be drawn

#### Throws

IllegalArgumentException	Issued when the size of the x array is different from that of the y array
NullPointerException	Issued when x or y is null

- **getBlueComponent**

***public int getBlueComponent()***

Returns the value of blue of the color currently specified

**Return Value**

Integer between 0 and 255

**Reference Item**

setColor(int, int, int)

- **getClipHeight**

***public int getClipHeight()***

Returns the height of a clipping rectangle

**Return Value**

Height of the clipping rectangle

**Reference Item**

clipRect(int, int, int, int), setClip(int, int, int, int)

- **getClipWidth**

***public int getClipWidth()***

Returns the width of a clipping rectangle

**Return Value**

Width of the clipping rectangle

**Reference Item**

clipRect(int, int, int, int), setClip(int, int, int, int)

- **getClipX**

***public int getClipX()***

Returns the x coordinate of a clipping rectangle in the graphics coordinate system

**Return Value**

X coordinate of a clipping rectangle

**Reference Item**

clipRect(int, int, int, int), setClip(int, int, int, int)

- **getClipY**

***public int getClipY()***

Returns the y coordinate of a clipping rectangle in the graphics coordinate system

**Return Value**

Y coordinate of a clipping rectangle

**Reference Item**

clipRect(int, int, int, int), setClip(int, int, int, int)

- **getColor**

***public int getColor()***

Returns the currently specified color

**Return Value**

Integer in the form of 0x00RRGGBB

**Reference Item**

setColor(int, int, int), setColor(int)

- **getFont**

***public Font getFont()***

Returns the currently specified font

**Return Value**

Currently specified font

**Reference Item**

Font, setFont(org.kwis.msp.lcdui.Font)

- **getGrayScale**

***public int getGrayScale()***

This method gets the value for the gray scale of the currently specified color. When the color is specified by setGrayScale, the value is returned as it is. When the color is specified as red, blue, or green, however, a value for the gray scale color closest to the color is returned.

**Return Value**

Integer between 0 and 255

**Reference Item**

setGrayScale(int)

- **getGreenComponent**

***public int getGreenComponent()***

Returns the value of green of the currently specified color

**Return Value**

Integer between 0 and 255

**Reference Item**

setColor(int, int, int)

- **getRedComponent**

***public int getRedComponent()***

Returns the value of red of the currently specified color

**Return Value**

Integer between 0 and 255

**Reference Item**

setColor(int, int, int)

- **getStrokeStyle**

***public int getStrokeStyle()***

Returns the currently specified stroke style used to draw lines, arcs, and rectangles

**Return Value**

Stroke style, i.e., SOLID or DOTTED

- **getTranslateX**

***public int getTranslateX()***

Returns the x coordinate of the starting point in the graphics coordinate system

**Return Value**

X coordinate of the starting point

**Reference Item**

getTranslateX(), translate(int, int)

- **getTranslateY**

***public int getTranslateY()***

Returns the y coordinate of the starting point in the graphics coordinate system

**Return Value**

Y coordinate of the starting point

**Reference Item**

getTranslateX(), translate(int, int)

- **setClip**

***public void setClip(int x, int y, int width, int height)***

This method specifies the area of a clipping rectangle. The drawing operation does not occur outside of the clipping area. The starting point of a rectangle starts in the current graphics coordinate system (x, y) with width and height.

**Parameters**

x	X coordinate of the starting point for the new clipping rectangle
y	Y coordinate of the starting point for the new clipping rectangle
width	Width of the new clipping rectangle
height	Height of the new clipping rectangle

**Throws**

IllegalArgumentException      Issued in case of negative width and height values

**Reference Item**

clipRect(int, int, int, int)

- **setColor**

***public void setColor(int rgb)***

This method specifies colors that are used in all painting operations. After this method is called, all painting operations should paint using colors specified by this method. Colors are passed on in RGB format like 0x00RRGGBB. The top-most 8 bits can have any value.

**Parameters**

rgb

- **setColor**

***public void setColor(int r, int g, int b)***

This method specifies colors that are used in all painting operations in RGB format. After this method is called, all painting operations should paint using colors specified by this method. Each color receives an RGB value separately.

**Parameters**

r	Value of red between 0 and 255
g	Value of green between 0 and 255
b	Value of blue between 0 and 255

**Throws**

IllegalArgumentException	Issued when values of r, g, and b are out of bounds
--------------------------	---

- **setFont**

***public void setFont(Font font)***

This method specifies the font used to draw a string. After this method is called, all drawing operations should draw strings using the font specified by this method. When the font is null, this method is the same as setFont(Font.getDefaultFont()).

**Parameters**

font	Font to be specified
------	----------------------

- **setGrayScale**

***public void setGrayScale(int val)***

All colors used in drawing operations should be specified in gray scale colors. After this method is called, all drawing operations should draw using the colors specified by this method. Drawings will be printed in the closest color that can be printed on the display. This value should be between 0 and 255.

**Parameters**

val	Gray scale color between 0 and 255
-----	------------------------------------

**Throws**

IllegalArgumentException	Issued when the gray scale colors are out of bounds
--------------------------	---

- **setStrokeStyle**

***public void setStrokeStyle(int style)***

This method determines the stroke style to be used in drawing. Painting and image drawing are not affected by this method.



**Parameters**

style    SOLID or DOTTED

**Throws**

IllegalArgumentException    Issued when the style does not have a valid value

- **translate**

***public void translate(int x, int y)***

This method moves the starting point of the graphics coordinate system. All coordinate systems related to the operation are changed by this method. After this method is called, all operations are affected.

**Parameters**

x        X coordinate of the new starting point (in the current coordinate system)

y        Y coordinate of the new starting point (in the current coordinate system)

**Reference Item**

getTranslateX(), getTranslateY()

- **getPixel**

***public int getPixel(int x, int y)***

This method brings pixels in a specific position in RGB format, i.e., pixel values from a specific (x, y) position of the display or an image in 0x00RRGGBB format. It is not affected by the clipping area.

**Parameters**

x        X coordinate of the pixel to be brought

y        Y coordinate of the pixel to be brought

**Return Value**

pixel value

**Throws**

IllegalArgumentException    Issued in case of negative width and height values

- **setPixel**

***public void setPixel(int x, int y)***

This method sets a pixel in a specific (x, y) position of the display or an image using the specified color. It is not affected by the clipping area.

**Parameters**

x      X coordinate of the pixel to be brought  
y      Y coordinate of the pixel to be brought

**Throws**

IllegalArgumentException      Issued when part of the target area is out of  
Graphics bounds or in case of negative width and  
height values

- **getPixels**

***public void getPixels(int x, int y, int w, int h, byte[] pixels, int offset, int bpl)***

This method brings the pixel values of the specific parts of the display or image related to the graphics. Data types of these pixels vary depending on the equipment, and they become identical with the data type specified in setPixel. This method is not affected by the clipping area.

**Parameters**

x	X coordinate of the pixel area to be brought
y	Y coordinate of the pixel area to be brought
w	Width of the pixel area to be brought
h	Height of the pixel area to be brought
pixels	Array where the pixel is to be stored
offset	Position where the pixel begins to be stored
bpl	Number of bytes required to store a line of images

**Throws**

ArrayIndexOutOfBoundsException	Issued when the array size of pixels is smaller than $(h * bpl + (w * \text{bit per pixel} + 7)/8)$
NullPointerException	Issued when pixels are null

- **getRGBPixels**

***public void getRGBPixels(int x, int y, int w, int h, int[] pixels, int offset, int bpl)***

This method brings the pixel values of the specific parts of the display or image related to the graphics. The data type of pixels shall be 0x00RRGGBB. This method is not affected by the clipping area.

**Parameters**

x	X coordinate of the pixel area to be brought
y	Y coordinate of the pixel area to be brought
w	Width of the pixel area to be brought
h	Height of the pixel area to be brought
pixels	Array where the pixel is to be stored
offset	Position where the pixel begins to be stored
bpl	Number of bytes required to store a line of images

**Throws**

ArrayIndexOutOfBoundsException	Issued when the array size of pixels is smaller than $(h * \text{bpl} + (w * \text{bit per pixel} + 7)/8)$
NullPointerException	Issued when pixels are null

- **setPixels**

***public void setPixels(int x, int y, int w, int h, byte[] pixels, int offset, int bpl)***

This method concurrently specifies the pixel values of the specific parts of the display or image related to the graphics. The data types of these pixels vary depending on the equipment, and they become identical with the data type specified in getPixel. This method is not affected by the clipping area.

**Parameters**

x	X coordinate of the pixel area to be brought
y	Y coordinate of the pixel area to be brought
w	Width of the pixel area to be brought
h	Height of the pixel area to be brought
pixels	Array where the pixel is to be stored
offset	Position where the pixel begins to be stored
bpl	Number of bytes required to store a line of images

**Throws**

ArrayIndexOutOfBoundsException	Issued when the array size of pixels is smaller than $(h * bpl + (w * \text{bit per pixel} + 7)/8)$
NullPointerException	Issued when pixels are null

- **setRGBPixels**

***public void setRGBPixels(int x, int y, int w, int h, int[] pixels, int offset, int bpl)***

This method concurrently specifies the pixel values of the specific parts of the display or an image related to the graphics. The data type of these pixels is 0x00RRGGBB, with each pixel stored in an integer.

**Parameters**

x	X coordinate of the pixel area to be brought
y	Y coordinate of the pixel area to be brought
w	Width of the pixel area to be brought
h	Height of the pixel area to be brought
pixels	Array where the pixel is to be stored
offset	Position where the pixel begins to be stored
bpl	Number of bytes required to store a line of images

**Throws**

ArrayIndexOutOfBoundsException	Issued when the array size of pixels is smaller than $(h * \text{bpl} + (w * \text{bit per pixel} + 7)/8)$
NullPointerException	Issued when pixels are null
IllegalArgumentException	Issued when part of the target area is out of Graphics bounds or in case of negative width and height values

This method is affected by the clipping area.

- **copyArea**

***public void copyArea(int dx, int dy, int sx, int sy, int w, int h)***

Copies a specific part of the display or an image related to graphics from one interior to another

**Parameters**

dx	X coordinate of the area of the position to be copied
dy	Y coordinate of the area of the position to be copied
sx	X coordinate of the area of the position to make a copy
sy	Y coordinate of the area of the position to make a copy
w	Width of the area to make a copy
h	Height of the area to make a copy

- **drawPolygon**

***public void drawPolygon(int[] x, int[] y)***

Draws a polygon using specified (x, y) points

**Parameters**

x	X array of the polygon to be drawn
y	Y array of the polygon to be drawn

**Throws**

IllegalArgumentException	Issued when the size of the x array is different from that of the y array
NullPointerException	Issued when x or y is null

- **reset**

***public void reset()***

Initializes all the contents contained in the graphics including style, clipping area, starting point of the coordinate system, and color font

- **setAlpha**

***public void setAlpha(int alpha)***

This method specifies the degree of transparency for all graphic operations. When the alpha value is 0, the drawing becomes transparent. On the other hand, when the value is 255, the drawing is printed on the display as it is. Other values will be regarded as 255. When transparency is specified, there will be no more drawing in XOR mode.

**Parameters**

alpha Degree of transparency to be specified

- **getAlpha**

***public int getAlpha()***

Brings the alpha value

**Return Value**

Specified alpha value

- **setXORMode**

***public void setXORMode(boolean b)***

Drawing is carried out in XOR mode. Using drawLine or drawPolygon, drawing will be carried out on the display in XOR mode. When changing to XOR mode, there will be no more transparent (alpha) drawing.

**Parameters**

B When True is returned, the drawing will be in XOR mode; otherwise, when False is returned, the drawing will be in common mode

- **isXORMode**

***public boolean isXORMode()***

Returns the set XOR mode

**Return Value**

True when the XOR mode is set

- **encodeImage**

***public byte[] encodeImage(int x, int y, int w, int h)***

This method encodes a specific area of the display in BMP format. The encoded BMP format is returned by byte array and can be stored as a file or used to generate images.

**Parameters**

x	Starting x coordinate of the area to be encoded
y	Starting y coordinate of the area to be encoded
w	Width of the area to be encoded
h	Height of the area to be encoded

**Throws**

IllegalArgumentException	Issued when part of the target area is out of Graphics bounds or in case of negative width and height values
--------------------------	--

**Return Value**

Byte array of encoded BMP format (in case encoding fails, a null value is returned)



**Class Image**

```
java.lang.Object
|
+--org.kwis.msp.lcdui.Image
```

```
public class Image extends Object
```

This is a class representing images. The image class can be generated from data of diverse image formats such as gif or png or by copying the existing images. In case of copied images, their content can be changed using getGraphics.

Images can be classified into those that are susceptible to change and those that are not susceptible. Images that are susceptible to change are mainly random images created in the programs. In contrast, images that are created from image files are not susceptible to change.

During copying, animation images do not lend themselves to image copying. In case of a masked image, the masked part is converted into white before it is copied. A copied image no longer has a mask.

The current image supports gif (animated), png, and bmp (including RLE compression) formats.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field**

- **TRAN\_ROT90**

***Public static final int TRAN\_ROT90***

Constant rotating the image to the right by 90° (specified as 90)

- **TRAN\_ROT180**

***Public static final int TRAN\_ROT180***

Constant rotating the image to the right by 180° (specified as 180)

- **TRAN\_ROT270**

***Public static final int TRAN\_ROT270***

Constant rotating the image to the right by 270° (specified as 270)

- **TRAN\_MIR**

***Public static final int MIR***

Constant indicating the right/left mirroring of the image (specified as -99)

- **TRAN\_MIR\_ROT90**

***Public static final int TRAN\_MIR\_ROT90***

Constant rotating the image to the right by 90° after right/left mirroring of the image (specified as -90)

- **TRAN\_MIR\_ROT180**

***Public static final int TRAN\_MIR\_ROT180***

Constant rotating the image to the right by 180° after right/left mirroring of the image (specified as -180)

- **TRAN\_MIR\_ROT270**

***Public static final int TRAN\_MIR\_ROT270***

Constant rotating the image to the right by 270° after right/left mirroring of the image (specified as -270)

### Detailed Description of the Generator

- **Image**

*protected Image()*

### Detailed Description of the Method

- **createImage**

***public static Image createImage(byte[] imagedata, int imageoffset, int imagelength)***

This method creates images from the content of the byte array specified by image data from imageoffset to imagelength. The content of the data may vary depending on the platform. `ArrayIndexOutOfBoundsException` is issued when imageoffset is smaller than 0 or larger than the length of image data (-1), when imagelength is smaller than 0, or when imageoffset + imagelength is larger than the length of imagedata. When imagelength is 0, `IllegalArgumentException` is issued.

Images created using this method cannot be edited.

**Parameters**

imagedata	Array with image data
imageoffset	Starting point of an array

imagelength    Length of image data

#### Return Value

Created image

#### Throws

IllegalArgumentException	Issued when the data is not in a proper format
NullPointerException	Issued when imagedata is null
ArrayIndexOutOfBoundsException	Issued when the area specified by imageoffset and imagelength are out of bounds for an array

#### ● loadImage

***public static Image loadImage(String str, ImageObserver io)***

This method reads images from data specified by a string, which is the path name of a relevant resource. This method reads data from this resource to create images. Images are created after a certain period of time has passed since this method is called. The notify method of ImageObserver is called at the time of completion.

An image loaded by this method initially has no content, the width and height are 0, and the appropriate width and height can be obtained after the image is actually completed (after ImageObserver.notify is called).

Only one ImageObserver is possible for an image.

Images created using this method cannot be edited.

#### Parameters

str	String that specifies the path name of the data
io	ImageObserver that indicates the creation of an image

#### Return Value

Newly created image; initially has no content

#### Throws

NullPointerException    Issued when str is null

#### Reference Item

ImageObserver.notify(org.kwis.msp.lcd.ui.Image, int)

- **createImage**

***public static Image createImage(Image image)***

This method copies a specified image to create another image that can be edited. A graphics object can be brought from this image through getGraphics() and edited. On the other hand, an animation image cannot be copied since there is a corresponding frame. In this case, this method throws IllegalArgumentException. Images created using this method can be edited.

**Parameters**

image Image to be copied

**Return Value**

Newly created image

**Throws**

IllegalArgumentException	Issued when the image passed on is an animation
NullPointerException	Issued when the image is null

- **createImage**

***public static Image createImage(int width, int height)***

This method creates an image with specified height and width that can be edited. This image is initialized in white as a default value. In addition, a graphics object can be obtained through the getGraphics() method, which can be used for image editing. Images created using this method can be edited.

**Parameters**

width	Width of the image
height	Height of the image

**Throws**

IllegalArgumentException	Issued when the width and height are 0 or less
--------------------------	--

- **createImage**

***public static Image createImage(String name) Throws IOException***

This method creates the image of a specified resource, loads a resource with a path name that is specified at the place where the relevant class is loaded, and creates an image to be returned. Images created using this method cannot be edited.

**Parameters**

name Path name of a resource

**Return Value**

Image

**Throws**

IllegalArgumentException	Issued when the image is in an invalid format, or the resource does not exist
NullPointerException	Issued when the name is null

**Reference Item**

Class.getResourceAsStream(java.lang.String)

- **getGraphics**

***public Graphics getGraphics()***

This method returns graphics that can draw on an image and gets a graphics object for an editable image. When the image is not editable, a null value is returned.

**Return Value**

Graphics object of an image

- **getHeight**

***public int getHeight()***

Returns the height of an image

**Return Value**

Height of the image

- **getWidth**

***public int getWidth()***

Returns the width of an image

**Return Value**

Width of the image

- **isMutable**

***public boolean isMutable()***

Indicates whether or not the image can be edited

**Return Value**

Whether or not editing is possible

- **isAnimated**

***public boolean isAnimated()***

Indicates whether or not animation is possible for the image

**Return Value**

Whether or not animation is possible

- **play**

***public void play(ImageObserver ob)***

This method starts the movement of an image. It is used when the image is an animation. During the implementation of animation, the status is disclosed to ImageObserver by calling the notify() method. When the image is not an animation image, this method does not play any role.

This method requires a considerable amount of computing time, because the image has to be decoded every time a new frame is created. Likewise, it consumes a considerable amount of memory since the content of the original image is stored as it is for animation image. When calling this method, the stop method also has to be called to ensure increased memory and speed. When ob is null, ImageObserver registered at loadImage() is used. If ImageObserver registered at loadImage() is different from ob, however, current ob is used.

**Parameters**

ob      ImageObserver

**Reference Item**

ImageObserver

- **stop**

***public void stop()***

This method stops an animation. It can be used when the image is an animation image, e.g., animation gif. When the image is not an animation image, this method does not play any role. This method should be called, and the image reference should be removed when the image is no longer used. Otherwise, memory may be wasted due to the unused image reference.

- **stopImage**

***public static void stopImage(ImageObserver io)***

This method stops reading an image corresponding to ImageObserver. It removes works piled up in "image reading queue" using loadImage or play. After deletion, ImageObserver.notify is no longer called. Application programs starting animation images using the play method should be stopped using the stop method or this method. Otherwise, CPU execution time and memory may be wasted since internal image decoding continues.

In case of a null io value, this method does not play any role.

**Parameters**

io      ImageObserver corresponding to the image to be removed

- **drawImage**

***public void drawImage(Image img, int srcX, int srcY, int srcWidth,  
int srcHeight, int destX, int destY, int transform, int anchor)***

This method draws a part of the image indicated by img in a specified position through conversion using Rotate/Flip. Conversion is carried out as shown in Table 3-1-1-2.

Table 3-1-1-2. Conversion-related constants

Constant	Description
TRAN_ROT90	Rotates 90° to the right
TRAN_ROT180	Rotates 180° to the right
TRAN_ROT270	Rotates 270° to the right
TRAN_MIR	Left-right mirroring
TRAN_MIR_ROT90	Rotates 90° to the right after left-right mirroring
TRAN_MIR_ROT180	Rotates 180° to the right after left-right mirroring
TRAN_MIR_ROT270	Rotates 270° to the right after left-right mirroring

**Parameters**

img	Original image
srcX, srcY	Coordinates of the original image that will become the starting point (0,0) of the sub-image
srcWidth, srcHeight	Width and height of the sub-image
destX, destY	X and y coordinates of the anchor point
transform	Specifies the conversion method
anchor	Anchor position of the text; see Anchor

**Throws**

IllegalArgumentException	Issued when the anchor value is not valid
NullPointerException	Issued when img is null

**Reference Item**

Image

● **createSubImage**

**Public Image createSubImage(int x, int y, int width, int height, boolean bMutable);**

Creates a sub-image

**Parameters**

x, y	Coordinates of the original image that will become the starting point (0,0) of the sub-image
width, height	Width and height of the sub-image
bMutable	Specified as 1 to make the image to be created editable; otherwise, it is specified as 0

**Throws**

IllegalArgumentException	Issued when x, y, width, or height are out of bounds
--------------------------	--



for the original image

**Reference Item**

Image

● **setTransparentColor**

***public void setTransparentColor(int rgb);***

Sets the specified color as transparent

**Parameters**

rgb    Colors to be specified as transparent (care should be taken when the image supports more than 256 colors)

**Reference Item**

Image

**Class Animatelmage**

java.lang.Object

|

+--org.kwis.msp.lcdui.**Animatelmage**

Among the images, some support animation images. For these images, an animation image class is added. Supported images include ABMP, SIS, and GIF, whereas supported formats are optional. In addition, this method can be used to combine the non-animated type of images to produce an animated effect.

**Detailed Description of the Generator**

- None

**Detailed Description of the Method**

- **createAnimatelmage**

***public static Animatelmage createAnimatelmage (String imageName, boolean loaded) Throws java.io.IOException***

Loads an animate image that can be accessed from Classpath to create an animate image; the image created using this method cannot be edited

**Parameters**

imageName	Name of the file to be loaded
loaded	Indicates whether or not to decode images from all frames when creating animatelmage; when set to true, images from all frames are decoded (when set to false, no decoding is performed)

**Throws**

NullPointerException Issued when the name is null

- **createAnimatelmage**

***public static Animatelmage createAnimatelmage(int frameNumber, int width, int height) Throws IllegalArgumentException***

Creates animation images with the same number of frames and similar width and height as specified; the image created using this method can be edited

**Parameters**

frameNumber	Number of frames
-------------	------------------

width	Width of the image to be created
height	Height of the image to be created

**Throws**

IllegalArgumentException      Issued when the parameter is 0 or out of bounds

- **isMutable**

***public boolean isMutable()***

Returns whether or not an image can be edited

**Return Value**

Whether or not an image is editable

- **setAnimationRate**

***public void setAnimationRate(int delay, int n)***

This method sets the animation delay time between n and n+1 frames. When n is smaller than 1 or larger than getMaxFrame(), animation delay time will not be set.

**Parameters**

delay Sets the delay time between frames in millisecond.

n Specifies the frame for setting the delay time, i.e., the delay time will be set between n and n+1 frames

**Throws**

IllegalArgumentException      Issued when n is smaller than 1 or larger than getMaxFrame() or in case of immutable image

- **getAnimationRate**

***public int setAnimationRate()***

Returns the animation speed

**Return Value**

Delay time between n and n+1 frames in millisecond

- **getMaxFrame**

***public int getMaxFrame()***

**Return Value**

Maximum number of frames for the relevant AnimatImage

- **getWidth**

***public int getWidth()***

Gets the width of an image frame

**Return Value**

Width of the frame

- **getHeight**

***public int getHeight()***

Gets the height of an image frame

**Return Value**

Height of the frame

- **getFrameImage**

***public Image getFrameImage(int frame) throws IllegalArgumentException***

Returns a desired frame image

**Return Value**

Null in case of error during decoding or immutable image

**Throws**

IllegalArgumentException	Issued when the frame value is smaller than 0 or larger than getMaxFrame()
--------------------------	--

- **setFrameImage**

***public void setFrameImage(Image im, int frame) throws  
IllegalArgumentException***

Inserts a mutable image in the relevant frame

**Parameters**

im	Image to be inserted
frame	Frame number to be inserted; 0<frame<=getMaxFrame

**Throws**

IllegalArgumentException	Issued when the size of the image to be inserted is different from that of AnimatImage or in case of a larger frame than getMaxFrame() or immutable image
--------------------------	---



- **play**

***public void play(org.kwis.msp.lcdui.Graphics g, int x, int y, int anchor, ImageObserver ob)***

This method activates the movement of an image. During animation, the status is disclosed to ImageObserver by calling the notify() method of ImageObserver. After calling this method, the stop method should be called. This API works with non-blocking. Without a frame image, it will not play. In case of any frame omitted in the process, apply the delay only without printing. A frame without delay set causes the default delay value to be set to 500 milliseconds.

**Parameters**

g        Graphics to be drawn  
x        X coordinate of the anchor point  
y        Y coordinate of the anchor point  
anchor   Anchor position of the text; see Anchor  
ob       Image Observer

**Throws**

NullPointerException        Issued when Graphics is null

- **stop**

***public void stop()***

This method stops an animation. When the image is no longer used, this method should be called, and the image reference should be removed. Otherwise, memory may be wasted due to the unused image reference.

- **stopImage**

***public static void stopImage(ImageObserver io)***

This method stops reading an image corresponding to ImageObserver and removes works piled up in "image reading queue" using play. After deletion, ImageObserver.notify is no longer called. Application programs starting animation images using the play method should be stopped using the stop method or this method. Otherwise, CPU execution time and memory may be wasted, since internal image decoding continues. In case of a null io value, this method does not play any role.

**Parameters**

io        ImageObserver corresponding to the image to be removed

- **paintFrame**

***public boolean paintFrame(Graphics g, int frame, int x, int y) throws  
NullPointerException, IllegalArgumentException***

Draws a frame corresponding to the frame in the x, y position of a graphics object

**Parameters**

g	Graphics to be drawn
frame	Number of the frame to be drawn
x	x coordinate to be drawn
y	y coordinate to be drawn

**Return Value**

True	Correctly displayed on the screen
False	Not displayed on the screen when this method is called during playing ()

**Throws**

NullPointerException	Issued when g is null
IllegalArgumentException	Issued when the method exceeds the bounds of 0 <= frame <= getMaxFrame()

- **setRepeat**

***public boolean setRepeat(Boolean isRepeat)***

Sets whether or not the animation is repeated; the default value is false

**Parameters**

IsRepeat	When True is returned, the animation repeats permanently; otherwise, when False is returned, the animation repeats only once
----------	---

**Return Value**

**True**

When repeated/not repeated is set correctly

**False**

When this method is called during playing () or in case of immutable image; the set value is ignored

- **isRepeat**

***public boolean isRepeat()***

Determines whether or not AnimationImage will be repeated

**Return Value**

True when AnimationImage is repeated; False when AnimationImage is played only once

- **getImageType**

***public String getImageType()***

Determines the type of AnimationImage

**Return Value**

Type of AnimationImage (represented by MIME), e.g., "image/gif" type in case of GIF image and "image/sis" type in case of SIS (in case of immutable image, a null value is returned)



**Class InputMethodHandler**

```

java.lang.Object
|
+--org.kwis.msp.lcdui.InputMethodHandler

```

public class **InputMethodHandler** extends Object

InputMethodHandler is in charge of handling characters that have been inputted by the user.

notifyKeyInput(int keyCode) is the method that handles user key inputs. Accordingly, it is necessary to call this method and pass on the currently inputted key values. In addition, receiving the characters handled following key inputs requires specifying a particular method called InputMethodListener. In the absence of the specified InputMethodListener, False is returned.

InputMethodListener can be specified through setInputMethodListener (org.kwis.msp.lcdui.InputMethodListener).

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

- **InputMethodHandler**

***public InputMethodHandler(int constraint)***

This method creates an instance of InputMethodHandler using given inputting constraints. The constraint receives a value defined by TextComponent. Among the modes currently supported by automata, the one that allows inputting depending on the constraint will be determined.

**Throws**

IllegalArgumentException      Issued when the constraint is not valid

**Detailed Description of the Method**

- **getCurrentModeCode**

***public String getCurrentModeCode()***

This method gets the standard language code corresponding to the current input mode. The standard language code is one that is specified by ISO 639. When a language distinguishes a capital letter and a small letter, "/S" or "/L" can be appended to each code. For example, the standard code for an English character is "EN," whereas that for an English small letter is "EN/S."

**Return Value**

Language code corresponding to the input mode

- **notifyKeyInput**

***public final boolean notifyKeyInput(int keyCode, int type)***

This method is called when key inputs should be handled in InputMethodHandler. It will handle characters corresponding to input key values under the current input mode and call notifyTextChanged of InputMethodListener registered through setInputMethodListener (org.kwis.msp.lcdui.InputMethodListener).

**Parameters**

KeyCode                      Input key value

**Return Value**

True when characters are handled based on key inputs; otherwise, False is returned

- **setInputMethodListener**

***public void setInputMethodListener(InputMethodListener imListener)***

This method specifies InputMethodListener, which will transmit characters that have been handled by InputMethodHandler based on key inputs. NotifyTextChanged of InputMethodListener is called to transmit the characters that have been inputted. Therefore, receiving characters that have been inputted requires specifying a particular InputMethodListener.

As null value is specified the currently registered InputMethodListener is removed.

**Parameters**

imListener              InputMethodListener; may also be a null value

- **changeCurrentModeToNext**

***public void changeCurrentModeToNext()***

This method calculates the next input mode based on the criteria of current mode in accordance with the currently specified constraint and changes the current input mode to the next calculated input mode.

- **getCurrentMode**

***public int getCurrentMode ()***

Gets the current input mode

**Return Value**

Input mode value

- **setCurrentMode**

***public boolean setCurrentMode(int mode)***

This method specifies the current input mode using the given mode value. When the given mode value is not supported by automata, `IllegalArgumentException` is issued.

**Parameters**

Mode    New input mode value to be specified

**Throws**

`IllegalArgumentException`      Issued when the given mode value is not supported  
by automata

- **hideSymbolCard**

***public void hideSymbolCard()***

Removes `CandidateWindow` from the current display

- **setSymbolPosition**

***public void setSymbolPosition(int x, int y, int w, int h)***

This method sets the position for printing special character cards on the display as well as its width and height when the current input mode of `InputMethodHandler` is `IM_SYMBOL`. In this case, values for `x`, `y`, `w`, and `h` should not be lower than 0. When a value below 0 is specified, `IllegalArgumentException` is issued.

**Parameters**

`x`      Value for the x coordinate  
`y`      Value for the y coordinate  
`w`      Value for width  
`h`      Value for height

**Throws**

`IllegalArgumentException`      Issued when the value for `x`, `y`, `w`, and `h` is "0" or  
less

**Class Jlet**

```
java.lang.Object  
|  
+--org.kwis.msp.lcdui.Jlet
```

```
public abstract class Jlet extends Object
```

This is an MSP application program.

All application programs using MSP should be prepared by inheriting Jlet. All resources of MSP will be managed as the resource for the application program in Jlet units. Threads and cards created by MSP will be removed from the system upon the termination of Jlet.

Jlet has three states. When Jlet is first generated, it automatically becomes active. When the program is temporarily stopped by the application program manager or by the user, it goes into a pause. From this state, Jlet can be reactivated by the application program manager or by the user.

Jlet can shift to “terminated” state from any of these states. In this case, however, Jlet should terminate the program.

For a shift in state, the program calls the pauseApp(), resumeApp(), startApp(), or terminatApp() method as the need arises as shown in Figure 3-1-1-1.



Figure 3-1-1-1. Shift of Jlet state

The program first calls the startApp method. The parameters that are passed on when the System.execute method is called are then returns.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field**

- **ACTIVE**

*public static final int ACTIVE*

- **DESTROYED**

*public static final int DESTROYED*

- **PAUSED**

*public static final int PAUSED***Detailed Description of the Generator**

- **Jlet**

*protected Jlet()*

Generates a new Jlet

**Detailed Description of the Method**

- **setActiveJlet**

*public static void setActiveJlet(Jlet ql)*

Activates the specified Jlet

- **getActiveJlet**

*public static Jlet getActiveJlet()*

This method gets the currently activated Jlet. In the absence of an activated Jlet, a null value is returned.

**Return Value**

Jlet being implemented on the upper level

- **getJletFromPID**

*public static Jlet getJletFromPID(int id)*

This method gets Jlet corresponding to a given ID. In the absence of a given Jlet, or in case of a wrong ID, a null value is returned.

**Return Value**

Jlet corresponding to the ID

- **getCurrentJlet**

***public static Jlet getCurrentJlet()***

Gets the currently implemented Jlet

**Return Value**

Jlet currently being implemented

- **getCurrentProgramID**

***public int getCurrentProgramID()***

This method returns the program ID that generated Jlet. The program ID is the only integer specified by the system.

**Parameters**

id      Program ID that generated Jlet

- **startApp**

***protected abstract void startApp(String[] args)***

This method is called when the program starts. Called after Jlet is generated, this method is capable of allocating the required system resources and putting the card on the display.

During implementation, this method is called only once. Arguments passed on to Jlet are passed on again as args. Args[0] represents the Jlet name, whereas args[1], args[2], etc., represent arguments passed on by the user.

**Parameters**

args      Arguments passed on by the user

- **pauseApp**

***protected void pauseApp()***

This method is called when stopping the program. This method is called by the system to request the application program to stop temporarily, although the program can also be stopped through interaction with the user. When the program is stopped, the system resources (network, serial, etc.) being used should be returned.

- **resumeApp**

***protected void resumeApp()***

This method is called when resuming the stopped program. It should be implemented in a manner that will enable the Jlet that has been suspended using the pauseApp method to be restarted and allow the re-allocation of system resources (network, serial ports, etc.) that had been passed on by pauseApp.

- **destroyApp**

***protected abstract void destroyApp(boolean unconditional)***

***throws org.kwis.msp.lcd.ui.JletStateChangeException***

This is the method that indicates program termination. Regardless of the state the program is in, it will be terminated when this method is called. The program should end in case True is returned unconditionally. When False is returned, the program can prevent termination by throwing JletStateChangeException depending on the situation. This method should return all the resources allocated by the program to the system and save important data. Nonetheless, care should be taken not to start an infinite loop if the unterminated state of the program continues.

**Parameters**

unconditional	When True is returned, the program is terminated unconditionally; otherwise, when False is returned, Jlet can throw JletStateChangeException to prevent program termination
---------------	---

**Throws**

org.kwis.msp.lcd.ui.JletStateChangeException

Issued when the program cannot be terminated under the present state; in case True is returned unconditionally, the program can be terminated even if it throws this exception

- **notifyDestroyed**

***public final void notify Destroyed ()***

This is a method used to terminate the Jlet application program. When calling this method, the program goes into a Terminated state to call the terminateApp method later. All the resources of the program can be returned by calling Jlet.terminateApp().

- **getAppProperty**

***public final String getAppProperty(String key)***

This method returns properties specified in each application program, i.e., returns a corresponding property string to the relevant key. In the absence of a corresponding property, a null value is returned.

**Parameters**

key      Key corresponding to the property to be found

**Return Value**

Corresponding property

- **getEventQueue**

***public final EventQueue getEventQueue()***

Returns an event queue related to Jlet

**Return Value**

Event queue



### 3.1.2. File

#### Class File

```
java.lang.Object
|
+--org.kwis.msp.io.File
```

All implemented interfaces: Connection, InputConnection, OutputConnection, StreamConnection

public class **File** extends Object and implements StreamConnection

This is a class that is used to support the basic methods of a file such as read and write as well as the stream method.

All filenames are represented by absolute paths. Actually, however, all files can be created and removed in a directory permitted by the platform. Users can use the absolute path regardless of the format supported by the platform.

In case of separators, this document follows the practice employed by the Unix system. Therefore, the use of "/" is allowed.

Similar to the FileSystem class, there are restrictions on the access method with regard to the method specifying the path. There are three access methods as shown below:

```
FileSystem.PRIVATE_ACCESS
FileSystem.SHARED_ACCESS
FileSystem.SYSTEM_ACCESS
```

To call the methods described below, the access level should be specified:

```
open(String, int, int)
```

Depending on the mode for opening, there are limits to the number of streams that can be opened.

When opening in Read-Only mode, one input stream can be opened; output stream cannot be opened, however.

When opening in Write-Only mode, one output stream can be opened; input stream cannot be opened, however.

When opening in Read/Write mode, output stream can be opened in each mode.

Among the file methods, the following are low-level APIs that ensure faster access:

```
read(byte[])
read(byte[], int, int)
write(byte[])
```

```

write(byte[], int, int)
write(int)
seek(int)
sizeof()

```

Using `openInputStream()`, `openOutputStream()`, `openDataInputStream()`, or `openDataOutputStream()`, faster access than reading and writing in a file is possible.

#### Methods inherited from class `java.lang.Object`

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

#### Detailed Description of the Field

- **maxInputStream**

***protected int maxInputStream***

Maximum number of `InputStream`s that can be opened

- **maxOutputStream**

***protected int maxOutputStream***

Maximum number of `OutputStream`s that can be opened

- **READ\_ONLY**

***public static final int READ\_ONLY***

Option for Read-only

- **WRITE**

***public static final int WRITE***

Option for starting writing from EOF (End of File) in case of an existing file

- **WRITE\_TRUNC**

***public static final int WRITE\_TRUNC***

Option for changing the file size to 0 prior to opening in case of an existing file

- **READ\_WRITE**

***public static final int READ\_WRITE***

Option for Read and Write

#### Detailed Description of the Generator

- **File**

***public File(String filename, int mode) throws IOException***

This method creates a specified file in the exclusive directory of the application program. For the filename, there is no difference between a file that starts with a separator ("/") and a file that starts with a filename without a separator. Created files may be opened in the following modes (the preferred mode can be specified, and the file can be opened as described below):

**READ\_ONLY** Only reading is allowed. When trying to write, an exception occurs.

**WRITE** The current file is maintained, and the file pointer is located at the very end.

**WRITE\_TRUNC** The file is truncated, i.e., upon opening, file length changes to 0.

**READ\_WRITE** Reading and writing can be carried out at the same time

When trying to open the file in a mode other than what has been described above, `IllegalArgumentException` is issued.

#### Parameters

filename	Absolute path of the file to be opened
mode	READ_ONLY, WRITE, WRITE_TRUNC, or READ_WRITE

#### Throws

`IOException` Issued when the file cannot be opened

#### Reference Item

`FileSystem`

#### ● File

***public File(String filename, int mode, int flag) throws IOException, SecurityException***

This method creates a file. For the filename, there is no difference between a file that starts with a separator ("/") and one that starts with a filename without a separator. Created files can be opened in the following modes:

**READ\_ONLY** Only Read is allowed. When trying to write, an exception occurs.

**WRITE** The current file is maintained, and the file pointer is located at the very end.

**WRITE\_TRUNC** The file is truncated, i.e., upon opening, file length changes to 0.

**READ\_WRITE** Read and Write can be carried out at the same time.

When trying to open the file in a mode other than what has been described above, `IllegalArgumentException` is issued

#### Parameters

filename	Absolute path of the file to be opened
mode	READ_ONLY, WRITE, WRITE_TRUNC, or READ_WRITE
flag	FileSystem.PRIVATE_ACCESS, FileSystem.SHARED_ACCESS, or FileSystem.SYSTEM_ACCESS

#### Throws

`IOException` Issued when the file cannot be opened

`SecurityException` Issued when trying to access a directory without authority

**Reference Item**

FileSystem

**Detailed Description of the Method**● **openInputStream*****public InputStream openInputStream() throws IOException***

Brings InputStream including DataInputStream; up to one InputStream can be opened

**Specified by**

openInputStream in Interface InputConnection

**Return Value**

InputStream for the file

**Throws**

IOException Issued when the file is not opened yet, or InputStream is already opened

**Reference Item**

InputStream

● **openDataInputStream*****public DataInputStream openDataInputStream() throws IOException***

Brings DataInputStream including InputStream; up to one DataInputStream can be opened

**Specified by**

openDataInputStream in Interface InputConnection

**Return Value**

DataInputStream for the file

**Throws**

IOException Issued when the file is not opened yet, or InputStream is already opened

**Reference Item**

DataInputStream

● **openOutputStream*****public OutputStream openOutputStream() throws IOException***

Brings OutputStream including DataOutputStream; up to one OutputStream can be

opened

**Specified by**

openOutputStream in Interface OutputConnection

**Return Value**

OutputStream for the file

**Throws**

IOException Issued when the file is not opened yet, or OutputStream is already opened

**Reference Item**

OutputStream

- **openDataOutputStream**

***public DataOutputStream openDataOutputStream() throws IOException***

Brings DataOutputStream including OutputStream; up to one DataOutputStream can be opened

**Specified by**

openDataOutputStream in Interface OutputConnection

**Return Value**

DataOutputStream for the file

**Throws**

IOException Issued when the file is not opened yet, or OutputStream is already opened

**Reference Item**

DataOutputStream

- **close**

***public void close() throws IOException***

Closes a file; an exception does not occur when a file is already closed

**Specified by**

close in Interface Connection

**Throws**

IOException Issued when the file cannot be closed properly

- **write**

***public int write(int b) throws IOException***

Used when only one byte is written on the file

**Parameters**

b      One byte to be written

**Return Value**

Number of actual bytes to be written in data

**Throws**

IOException      Issued when writing in a file that is closed using the close method, or if it is not possible to write properly

- **write**

***public int write(byte[] buf) throws IOException, NullPointerException***

This method writes the largest possible volume of data that can be contained in the buf lengthwise in a file. When the buf is null, NullPointerException is issued.

**Parameters**

buf      Byte array that contains actual data

**Return Value**

Number of actual bytes to be written in data

**Throws**

IOException      Issued when writing in a file that is closed using the close Method, or if it is not possible to write properly  
NullPointerException      When the buf is null

- **write**

***public int write(byte[] buf, int off, int len) throws IOException, NullPointerException***

Writes the full len of data contained in the buf from off in a file (when the buf is null, NullPointerException is issued)

**Parameters**

buf      Byte array that contains actual data  
off      Position where the data is to be written  
len      Size of the actual data to be written

**Return Value**

Number of actual bytes to be written in data

**Throws**

IOException Issued when writing in a file that is closed using the close  
Method, or if it is not possible to write properly

NullPointerException When the buf is null

- **read**

***public int read() throws IOException***

Reads 1 byte from the input stream; in case of empty EOF, -1 is returned

**Return Value**

Number of bytes read; in case of 0 bytes, -1 is returned

**Throws**

IOException Issued in case of error during reading

- **read**

***public int read(byte[] buf) throws IOException, NullPointerException***

This method reads as much data as the buf size from the input stream. When the full length of the file is read, the same volume of data is stored in the buf, returning the same volume of data stored. When reading data next time, -1 is returned.

**Parameters**

buf Byte array that contains read data

**Return Value**

Number of bytes read or -1 when EOF is encountered before a single byte is read

**Throws**

IOException Issued when writing in a file that is closed using the close  
Method, or if it is not possible to write properly

NullPointerException Issued when the buf is null

- **read**

***public int read(byte[] buf, int off, int len) throws IOException,  
NullPointerException***

***IndexOutOfBoundsException***

This method reads as much len byte of data from the input stream. When EOF is

encountered before reading as much len byte of data, the same volume of data that is read is stored in the buf and returned.

#### Parameters

buf     Byte array that contains read data  
 off     Offset that determines data to be stored in a part read in buf  
 len     Represents how much is to be read

#### Return Value

Number of bytes read or -1 when EOF is encountered before a single byte is read

#### Throws

IOException                      Issued when writing in a file that is closed using the close Method, or if it is not possible to write properly  
 NullPointerException          Issued when the buf is null  
 IndexOutOfBoundsException      Issued when len is 0 or less, offset value is negative, or value of offset + len is out of bounds of buf

#### ● seek

***public void seek(int pos) throws IOException, IllegalArgumentException***

Moves the file pointer to a specific position

#### Parameters

pos     Pointer position of the file to be moved (should be an absolute value from the beginning of the file)

#### Throws

IOException                      Issued when the file handle is not set properly or in case of an error while moving the file pointer  
 IllegalIOException              Issued in case of invalid or negative pos value

#### ● sizeOf

***public int sizeOf() Throws IOException***

Indicates the file size

#### Return Value

File size

#### Throws

IOException                      Issued when the file handle is not set properly or in case of



an error while reading the file size

- **tell**

***public int tell() Throws IOException***

Gets the current position of the file pointer

**Return Value**

Offset value from the beginning of the file to the current position

**Throws**

IOException	Issued when the file handle is not set properly or in case of an error while reading the file pointer
-------------	---

**Class FileSystem**

```
java.lang.Object
|
+--org.kwis.msp.io.FileSystem
```

public final class **FileSystem** extends Object

The **FileSystem** class defines common methods related to a file such as create, delete, and change name. Paths for all files and directories are absolute paths. Users cannot specify these absolute paths, however.

The paths can be specified in three ways as described below (the system specifies proper absolute paths accordingly):

There are three types of access to a file:

For access to a directory that is used exclusively by the application program itself, the **PRIVATE\_ACCESS** flag is used. For access to a directory that is shared with other programs, **SHARED\_ACCESS** is used. Finally, for access to a system directory, the **SYSTEM\_ACCESS** type is used.

All methods that specify paths should specify an appropriate access method. What follows is a list of these access methods:

```
exists(java.lang.String, int)
isDirectory(java.lang.String, int)
isFile(java.lang.String, int)
list(java.lang.String, int)
mkdir(java.lang.String, int)
rmdir(java.lang.String, int)
remove(java.lang.String, int)
rename(java.lang.String, int)
```

If the application program wants to search for any file called test in its own directory, the following action is taken:

```
if (exists("test," PRIVATE_ACCESS)){
    System.err.println("test exists");
};
```

Unless specified as follows, the default value will be used in searching the program's own directory:

```
if (exists("test")){
    System.err.println("test exists");
};
```

As described above, when trying to search for any file called test, the filename will be found in the absolute path created by the system.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field**

- **PRIVATE\_ACCESS**

***public static final int PRIVATE\_ACCESS***

This is the flag used to access a directory exclusively reserved for the application program. This method is defined as 1.

- **SHARED\_ACCESS**

***public static final int SHARED\_ACCESS***

This is a flag used to access a directory shared with other programs. The directory to be shared is specified in the Jlet Descriptor when the program is installed, and the user cannot change the shared directory. This method is defined as 2.

- **SYSTEM\_ACCESS**

***public static final int SYSTEM\_ACCESS***

This is a flag that is used to access a directory used by the application program of the system. The directory to be shared is specified in the Jlet Descriptor when the program is installed, and the user cannot change the shared directory. This method is defined as 3.

- **MAX\_FILENAME\_LENGTH**

***public static final int MAX\_FILENAME\_LENGTH***

There is a limit to the length of a filename.

In all APIs containing a filename, IOException is issued when a filename is longer than MAX\_FILENAME\_LENGTH.

**Detailed Description of the Generator**

- **FileSystem**

***public FileSystem()***

**Detailed Description of the Method**

- **getMaxFilenameLength**

***public static int getMaxFilenameLength()***

Verifies the maximum length of a filename that can be used

**Return Value**

Maximum length of a filename

- **list**

***public static Vector list(String dirname) throws IOException***

This method shows all files and directories under a directory that is reserved exclusively for the application program itself. When the dirname is null, NullPointerException is issued. On the other hand, when the dirname is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

dirname	Directory name
---------	----------------

**Return Value**

All filenames and directory names under dirname

**Throws**

IOException	Issued in the absence of a directory
-------------	--------------------------------------

- **list**

***public static Vector list(String dirname, int flag) throws IOException, SecurityException***

This method shows all files and directories in a specified directory. When the dirname is null, NullPointerException is issued. When the dirname is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

dirname	Directory name
flag	Directory to be accessed

**Return Value**

All filenames and directory names under dirname

**Throws**

SecurityException	Issued when trying to access a directory without authorization
IOException	Issued when there is no dirname

- **exists**

***public static boolean exists(String name) throws IOException***

This method verifies if there is a file or a directory under the directory that is reserved exclusively for the application program itself. When the name is null, NullPointerException is issued. When the name is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

name	File or directory name
------	------------------------

**Return Value**

True when the file or directory exists; otherwise, False is returned

- **exists**

***public static boolean exists(String name, int flag) throws IOException, SecurityException***

This method verifies if there is a file or directory in a specified directory. When the name is null, NullPointerException is issued. When the name is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

name	Filename or directory name
flag	Directory to be accessed

**Return Value**

True when the file or directory exists; otherwise, False is returned

**Throws**

SecurityException	Issued when trying to access a directory without authorization
-------------------	--

- **remove**

***public static void remove(String filename) throws IOException***

This method removes a file in the directory reserved exclusively for the application program itself. When the filename is null, NullPointerException is issued. When the filename is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

filename	Filename
----------	----------

**Throws**

IOException	Issued when the file is not removed properly, or it does not exist
-------------	--

- **remove**

***public static void remove(String filename, int flag) throws IOException, SecurityException***

This method removes a file. When the filename is null, NullPointerException is issued. When the file does not exist, an exception does not occur. When the filename is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

filename	Filename
flag	Directory to be accessed

**Throws**

SecurityException	Issued when trying to access a directory without authorization
IOException	Issued when the file is not removed properly, or it does not exist

- **mkdir**

***public static void mkdir(String dirname) throws IOException***

This method creates a directory under the directory that is reserved exclusively for the application program itself. When the dirname is null, NullPointerException is issued. When the dirname is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

dirname	Directory name
---------	----------------

**Throws**

IOException	Issued when the directory cannot be made, it already exists,
-------------	--

or the length of the filename exceeds the maximum  
filename length

- **mkdir**

***public static void mkdir(String dirname, int flag) throws IOException,  
SecurityException***

This method makes a directory. When the dirname is null, NullPointerException is issued. When the dirname is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

dirname	Directory name
flag	Directory to be accessed

**Throws**

SecurityException	Issued when trying to access a directory without authorization
IOException	Issued when the directory cannot be made, it already exists, or the filename is longer than the maximum filename length

- **rmdir**

***public static void rmdir(String dirname) throws IOException***

This method removes a directory under the directory reserved exclusively for the application program itself. When the dirname is null, NullPointerException is issued. When the dirname is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

dirname	Directory name
---------	----------------

**Throws**

IOException	Issued when the directory is not empty, it cannot be removed, or it does not exist
-------------	--

- **rmdir**

***public static void rmdir(String dirname, int flag) throws IOException,  
SecurityException***

This method removes a directory. When the dirname is null, NullPointerException is issued. When the dirname is longer than MAX\_FILENAME\_LENGTH, IOException is issued.

**Parameters**

dirname	Directory name
---------	----------------

flag                      Directory to be accessed

#### Throws

SecurityException	Issued when trying to access a directory without authorization
IOException	Issued when the directory is not empty, it cannot be removed, or it does not exist

#### ● toCString

***public static byte[] toCString(String jStr)***

This method replaces Java String with C String. Encoding methods include ISO8859 (for English) and KSC5601 (for Korean).

#### Parameters

jStr                      java String

#### Return Value

Byte array replacing jStr with C string

#### Throws

NullPointerException	Issued when jStr is null
----------------------	--------------------------

#### ● available

***public static int available()***

Discloses the available space for the application program

#### Return Value

Available space in the file system

#### ● isFile

***public static boolean isFile(String name) throws java.io.IOException***

This method verifies if there is a file with the specified name in the directory that is reserved exclusively for the application program itself. A file or directory with the same name cannot exist under a directory with the same name.

#### Parameters

name    Filename to be verified

#### Return Value

True when there is a file with the specified name; otherwise, False is returned when it is a directory name, or such file does not exist



**Throws**

Java.io.IOException	Issued in case of longer name than MAX FILENAME LENGTH
NullPointerException	Issued when name is null

● **isFile**

***public static boolean isFile(String name, int flag) throws IOException,  
SecurityException***

This method verifies if there is a file with the specified name. A file or directory with the same name cannot exist under a directory with the same name.

**Parameters**

name	Filename to be verified
flag	Directory to be accessed

**Return Value**

True when there is a file with the specified name; otherwise, False is returned when it is a directory name, or such file does not exist

**Throws**

NullPointerException	Issued when name is null
IOException	Issued in case of longer name than MAX FILENAME LENGTH
SecurityException	Issued when trying to access a directory without authorization

● **isDirectory**

***public static boolean isDirectory(String name) throws IOException***

This method verifies if there is a directory with the specified name under the directory that is reserved exclusively for the application program itself. A file or directory with the same name cannot exist under a directory with the same name.

**Parameters**

name	Directory name to be verified
------	-------------------------------

**Return Value**

True when there is a directory with the specified name; otherwise, False is returned when it is a filename, or such directory does not exist

● **isDirectory**

***public static boolean isDirectory(String name, int flag) throws IOException, SecurityException***

This method verifies if there is a directory with the specified name. A file or directory having the same name cannot exist under a directory with the same name.

#### Parameters

name    Directory name to be verified  
flag    Directory to be accessed

#### Return Value

True when there is a directory with the specified name; otherwise, False is returned when it is a filename, or such directory does not exist

#### Throws

NullPointerException	Issued when name is null
IOException	Issued in case of longer name than MAX FILENAME LENGTH
SecurityException	Issued when trying to access a directory without authorization

#### ● getCreationTime

***public static int getCreationTime(String name) throws IOException***

This method gets the creation time for a file in the directory reserved exclusively for the application program itself. In case the name is a directory name, -1 is returned.

#### Parameters

name    Filename to be verified

#### Return Value

When the name is a directory name, -1 is returned; otherwise, when the name is a filename, it is represented by the creation time in second

#### Throws

NullPointerException	Issued when name is null
IOException	Issued in case of longer name than MAX FILENAME LENGTH

#### ● getCreationTime

***public static int getCreationTime(String name, int flag) throws IOException, SecurityException***

This method gets the creation time for a file. In case the name is a directory name, -1 is returned.

#### Parameters

name    Filename to be verified

#### Return Value

When the name is a directory name, -1 is returned; otherwise, when the name is a filename, it is represented by the creation time in second

#### Throws

NullPointerException	Issued when name is null
IOException	Issued in case of longer name than MAX FILENAME LENGTH

#### ● rename

***public static void rename(String oldName, String newName) throws  
IOException***

The application program changes the names of the files in a personal directory and a child directory. In the case of a child directory, the path of the parent directory should be included in the filename. The directory for the oldName file and the newName file should be the same. When oldName and newName are null, NullPointerException is issued. When oldName and newName are longer than MAX\_FILENAME\_LENGTH, IOException is issued.

#### Parameters

oldName	Current name
newName	Changed name

#### Throws

IOException	Issued in case of an error in the process of change
-------------	---

#### ● rename

***public static void rename(String oldName, String newName, int flag) throws  
IOException, SecurityException***

This method changes a filename. The current name of the file to be changed and the changed name cannot be used as paths that include a directory, i.e., a directory separator cannot be included.

**Parameters**

oldName	Current name
newName	Changed name
flag	Directory to be accessed

**Throws**

NullPointerException	Issued when newName is null
SecurityException	Issued when trying to access a directory without authorization
IOException	Issued in case of longer oldName and newName than MAX FILENAME LENGTH

### 3.1.3. Database

#### Interface DataComparator

All known implementing classes: DataComparatorInteger, DataComparatorString

public interface **DataComparator**

This is an interface that is used when comparing two records. This interface is necessary to sort the database. The sortRecord method determines the sequence of records in the database using the compare method of this interface. To sort the records, the application should implement this interface. For the comparison of basic data types such as integer and string, classes already implemented are provided.

#### Detailed Description of the Field

- **EQUIVALENT**

*public static final int EQUIVALENT*

Two records received as parameters during record sorting or search are equivalent in terms of sequence.

- **FOLLOWS**

*public static final int FOLLOWS*

The first record received as a parameter during record sorting or search comes after the second record.

- **PRECEDES**

*public static final int PRECEDES*

The second record received as a parameter during record sorting or search comes after the first record.

#### Detailed Description of the Method

- **compare**

*public int compare(byte[] data1, byte[] data2)*

This is a method (comparator) used to compare records. In implementing this method, note that the byte arrays received as parameters should follow the format of the record data stored in the database.

**Parameters**

data1    Record data to be compared

data2    Record data to be compared

**Return Value**

DataComparator.EQUIVALENT when two records are equivalent in sequence;

DataComparator.FOLLOWS when data2 is followed by data1 (i.e., data1 follows

2); DataComparator.PRECEDES when data1 is followed by data2

**Interface DataFilter**

All known implementing classes: DataFilterInteger

public interface **DataFilter**

This interface limits the records to be used for sorting and determines whether or not to include certain records in the sorting when sorting database records using the sortRecord method. For DataFilter with basic data types such as integer and string, classes already implemented are provided.

**Detailed Description of the Method**

- **filter**

***public boolean filter(byte[] data)***

This is a method that limits records to be used for sorting and determines whether or not to include certain records in the sorting. In implementing this method, note that the byte arrays received as parameters should follow the format of the record data stored in the database.

**Parameters**

data    Byte array representing data stored in a record

**Return Value**

True when the record is included in sorting; otherwise, False is returned

**Class DataBase**

```
java.lang.Object
|
+--org.kwis.msp.db.DataBase
```

```
public class DataBase extends Object
```

This class provides a mechanism for the storage, search, and management of perpetual data. It is used to implement a simple database in Jlet. Multiple records (data) can be stored and read in a database.

Actual records are stored in the perpetual area of the platform to guarantee the minimum integrity of the stored records (i.e., if the platform goes down while the record is being stored, the integrity of records cannot be guaranteed) even when Jlet implementation is completed, or the platform is down.

Records are stored in the form of byte arrays. The meaning of data to be stored is ignored. The user is responsible for grasping the logical meaning of stored data. The size of data cannot exceed the specified size when the database is first created.

Each record is represented as an integer value called record ID in the database. Record ID starts from 0, increasing by 1 each time a record is stored unless the records are deleted. When a record is deleted in the middle, the deleted record ID is re-used when storing the next record.

The frequent addition and deletion of records may result in a difference between the number of records stored in the database and the capacity of the database in the file system of the actual platform. MSP does not provide a mechanism (compaction) that makes use of this kind of empty space.

One Jlet can create a number of databases and can access all the databases created by its own Jlet. In most cases, however, it cannot access a database created by another Jlet.

A database can be shared by several Jlets by creating it in a shared directory; otherwise, the database used by the system application can be accessed by awarding a flag when the database is opened.

Deleting Jlet on the platform also deletes the database as well as the resources (e.g., files) created by the database in the physical area of the platform.

To sort the database, the `DataFilter` interface and `DataComparator` interface should be implemented. The `DataFilter` interface selects records that are necessary for sorting, whereas the `DataComparator` interface compares two records for sorting.



**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Method**

- **openDataBase**

***public static DataBase openDataBase(String dataBaseName, int recordSize, boolean create) throws DataBaseException, IllegalArgumentException***

This method opens a database that can be accessed only through the current Jlet. The database is opened in FileSystem.PRIVATE\_ACCESS mode.

**Parameters**

dataBaseName	Database name
recordSize	Size of one record (in bytes) of the database to be created; when a database already exists, the specified recordSize is ignored, and the existing record is applied
create	Whether or not a new database is to be created (in case a database does not exist)

**Return Value**

Open database

**Throws**

DataBaseException	Issued when create is False; in the absence of a database, a file used in the database cannot be opened, or the database is broken
IllegalArgumentException	Issued when create is True, and if recordSize is 0 or a negative value

- **openDataBase**

***public static DataBase openDataBase(String dataBaseName, int recordSize, boolean create, int flag) throws DataBaseException, IllegalArgumentException***

This method opens a database. Using a flag enables specifying issues such as whether or not access will be limited to Jlet, whether or not a database shared with other Jlets will be opened, or whether or not the database provided by the system will be used.

**Parameters**

dataBaseName	Database name
--------------	---------------

recordSize	Size of one record (in bytes) of the database to be created; when a database already exists, the specified recordSize is ignored, and the existing record is applied
create	Whether or not a new database is to be created (in case a database does not exist)
flag	Specifies a sharing method of the database from among FileSystem.PRIVATE_ACCESS, FileSystem.SHARED_ACCESS, and FileSystem.SYSTEM_ACCESS

**Return Value**

Open database

**Throws**

DataBaseException	Issued when create is False; in the absence of a database, a file used in the database cannot be opened, or the database is broken
IllegalArgumentException	Issued when create is True, and if recordSize is 0 or a negative value

● **closeDataBase*****public void closeDataBase() throws DataBaseException***

Closes a database; ignored when closing an already closed database

**Throws**

DataBaseException	Issued when database-related information cannot be stored, or the file cannot be closed
-------------------	---

● **deleteDataBase*****public static void deleteDataBase(String dataBaseName) throws DataBaseException***

Deletes only the database created by Jlet using FileSystem.PRIVATE\_ACCESS

**Parameters**

dataBaseName Name of the database to be deleted

**Throws**

DataBaseException	Issued in the absence of a database to be deleted, or if the database cannot be deleted
-------------------	---



- **deleteDataBase**

***public static void deleteDataBase(String dataBaseName, int flag)***

***throws DataBaseException, IllegalArgumentException***

Deletes only the database that can be accessed by Jlet with the flag

**Parameters**

dataBaseName	Name of the database to be deleted
flag	Represents the database to be deleted in terms of access right classification such as FileSystem.PRIVATE_ACCESS, FileSystem.SHARED_ACCESS, and FileSystem.SYSTEM_ACCESS

**Throws**

DataBaseException	Issued in the absence of a database to be deleted, or if the database cannot be deleted
IllegalArgumentException	Issued in case of incorrect flag value

- **insertRecord**

***public int insertRecord(byte[] data, int offset, int numBytes) throws***

***DataBaseRecordException, DataBaseException, IllegalArgumentException***

This method adds a new record to the database. Data in the byte array is stored as a record in the database. The data is written in the physical area of the platform before it is returned. When the byte array to be stored is shorter than the size of a specified record when the database is created, garbage can be stored in the remaining area. Since selectRecord reads data in record size as a unit, the user is responsible for distinguishing the actual data and the garbage from the byte array that is read again after it is stored.

**Parameters**

data	Buffer containing the data to be stored
offset	First byte offset where the data to be stored in the buffer starts
numBytes	Number of bytes to be stored

**Return Value**

Record ID of the stored record

**Throws**

DataBaseRecordException	Issued when the data is larger than the specified record size when the database is created
-------------------------	--

DataBaseException	Issued when the record cannot be stored
IllegalArgumentException	Issued when the value of the data length after deduction of offset value is smaller than numBytes

- **insertRecord**

***public int insertRecord(byte[] data) throws DataBaseRecordException, DataBaseException***

This method adds a new record to the database. Data in the byte array is stored as a record in the database. The data is written in the physical area of the platform before it is returned. When the byte array to be stored is shorter than the size of a specified record when the database is created, garbage can be stored in the remaining area. Since selectRecord reads data in record size as a unit, the user is responsible for distinguishing the actual data and the garbage from the byte array that is read again after it is stored.

**Parameters**

data     Buffer containing the data to be stored

**Return Value**

Record ID of the stored record

**Throws**

DataBaseRecordException	Issued when the data is larger than the specified record size when the database is created
DataBaseException	Issued when the record cannot be stored

- **deleteRecord**

***public void deleteRecord(int recordId) throws DataBaseException, DataBaseRecordException***

Deletes a record from the database corresponding to the Record ID received as a parameter

**Parameters**

recordId     Record ID to be deleted

**Throws**

DataBaseException	Issued when the record cannot be deleted
DataBaseRecordException	Issued in the absence of a record ID

- **selectRecord**

***public byte[] selectRecord(int recordId) throws DataBaseException,***

***DataBaseRecordException***

Returns the stored data to a specific record ID

**Parameters**

recordId	Record ID
----------	-----------

**Return Value**

Data stored in the record ID

**Throws**

DataBaseException	Issued when the record cannot be read
DataBaseRecordException	Issued in the absence of a record ID

- **selectRecord**

***public void selectRecord(int recordId, byte[] buffer, int offset) throws***

***DataBaseException, DataBaseRecordException, IllegalArgumentException***

This method returns the stored data to a specific record ID. Read data is copied to the buffer and returned. When the content of the buffer is changed later, the record stored in the database does not change. The buffer should be large enough to contain a record. The size of a record in the database can be obtained using the getRecordSize method.

**Parameters**

recordId	Record ID
buffer	Buffer where the read data is copied and stored
offset	First byte offset where copying starts in the buffer

**Throws**

IllegalArgumentException	Issued when the buffer is smaller than the record size
DataBaseException	Issued when the record cannot be read
DataBaseRecordException	Issued in the absence of a record ID

- **updateRecord**

***public void updateRecord(int recordId, byte[] newData, int offset, int numBytes) throws DataBaseRecordException, DataBaseException, IllegalArgumentException***

Changes the content of a specific record stored in a record ID into a new content

**Parameters**

recordId	Record ID whose content is to be changed
newData	Buffer containing the data for storage

offset	First byte offset where copying starts in the buffer
numBytes	Number of bytes to be stored

**Throws**

DataBaseException	Issued when the data cannot be stored
DataBaseRecordException	Issued when the data is larger than the specified record size when the database is created, or in the absence of a record ID
IllegalArgumentException	Issued when the value of the buffer length after deducting offset is smaller than the number of bytes to be stored

- **updateRecord**

***public void updateRecord(int recordId, byte[] newData)***  
***throws DataBaseException, DataBaseRecordException***

Changes the content of a specific record stored in a record ID into a new content

**Parameters**

recordId	Record ID whose content is to be changed
newData	Buffer containing the data for storage

**Throws**

DataBaseException	Issued when the data cannot be stored
DataBaseRecordException	Issued when the data is larger than the specified record size when the database is created, or in the absence of a record ID

- **sortRecord**

***public int[] sortRecord(DataFilter filter, DataComparator comparator) throws DataBaseException***

This method sorts records in accordance with the comparison method and the defined limit condition.

The record comparison method can be defined by implementing the DataComparator interface. DataComparator is a class that compares two records. When the comparator is null, unsorted record IDs are returned.

The record limit condition can be defined by implementing the DataFilter interface, and only records satisfying the condition are used in sorting. When the filter is null, the limit condition is regarded as non-existent, and all records of the database are used in sorting. In short, when only the filter is null, all records are sorted by the specified comparator, and their record IDs are returned. When only the comparator is null, the record IDs of

records satisfying the conditions specified by the filter are returned. When both the filter and the comparator are null, all record IDs stored in the database are returned without sorting.

For conditions of integer scope and string, pre-implemented classes (`DataFilterInteger`, `DataComparatorInteger`) can be used.

This method returns integer arrays with record IDs. To get each record, the `selectRecord` method can be used using the obtained record IDs.

**Parameters**

filter	Record limit condition; the condition can be null
comparator	Record comparison method; the method cannot be null

**Return Value**

Null value in the absence of an array record for the sorted record ID

**Throws**

`DataBaseException` Issued when the records cannot be sorted

**● listDataBases*****public static String[] listDataBases()***

This method returns an array of database names. Jlet returns all names of the databases that are created by `FileSystem.PRIVATE_ACCESS` and names of databases that can be accessed from among those databases created by `FileSystem.SHARED_ACCESS` and `FileSystem.SYSTEM_ACCESS`.

**Return Value**

Names of databases or null value in the absence of databases

**● getAccessMode*****public static int getAccessMode(String dbName) throws DataBaseException***

Returns the access right to a database

**Parameters**

dbName	Database name indicating the access right
--------	---

**Return Value**

`FileSystem.PRIVATE_ACCESS`, `FileSystem.SHARED_ACCESS`, or  
`FileSystem.SYSTEM_ACCESS`

**Throws**

`DataBaseException` Issued in the absence of a database, or if the `dbName` is null





- **getDataBaseName**

***public String getDataBaseName()***

Returns the name of the database of an open instance

**Return Value**

Name of the database

- **getDataBaseSize**

***public int getDataBaseSize()***

This method returns the size of a database. The return value includes the size of the record stored in the database as well as the necessary size for the storage and management of the record. Therefore, the value to be returned may differ from the value obtained from the multiplication of the number of actual records stored in the database by the size of one record (a larger value is usually returned).

**Return Value**

Physical size of the database

- **getNumberOfRecords**

***public int getNumberOfRecords()***

Returns the number of records stored in a database

**Return Value**

Number of records

- **getRecordSize**

***public int getRecordSize()***

This method returns the size of one record stored in a database. Note that this does not include the required overhead to store the record.

**Return Value**

Size (byte) of one record

- **getSizeAvailable**

***public int getSizeAvailable()***

This method returns the size of the available capacity for future storage. Since the record to be stored requires an overhead for management and storage, the data may not be stored even if its actual size is smaller than the available capacity, which varies

depending on the size of the hardware.

**Return Value**

Available capacity (byte)

- **getLastModified**

***public long getLastModified()***

This method returns the time when the database is most recently modified. The format follows the one returned by `System.currentTimeMillis()`.

**Return Value**

Time when the database is most recently modified

**Class DataComparatorInteger**

```

java.lang.Object
|
+--org.kwis.msp.db.DataComparatorInteger

```

All implemented interfaces: DataComparator

public class **DataComparatorInteger** extends Object implementing DataComparator

This is a class that compares two records using integers. It is used to call the sortRecord method of the database.

**Fields inherited from the interface org.kwis.msp.db.DataComparator**

EQUIVALENT, FOLLOWS, PRECEDES

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

- **DataComparatorInteger**

***public DataComparatorInteger(int offset)***

This is a class that compares two records using integers.

When calling the sortRecord method in a database, the class that implemented the DataComparator interface can pass on this class or DataComparatorInteger. sortRecord will then sort records considering the four bytes, starting with the offset that is passed on when this class is created as an integer. The byte order is based on big-endian.

This is required when calling sortRecord.

**Parameters**

offset    Byte offset where the integer to be compared starts in the record

**Detailed Description of the Method**

- **compare**

***public int compare(byte[] data1, byte[] data2)***

This is a method (comparator) that compares records. In implementing this method, note that the byte arrays received as parameters should follow the format of the record data stored in the database.

**Parameters**

data1    Record data to be compared

data2    Record data to be compared

**Return Value**

DataComparator.EQUIVALENT when two records are equivalent in sequence;  
DataComparator.FOLLOWS when data2 is followed by data1 (i.e., data1 follows  
2); DataComparator.PRECEDES when data1 is followed by data2

**Class DataComparatorString**

java.lang.Object

|

+--org.kwis.msp.db.DataComparatorString

All implemented interfaces: DataComparator

public class **DataComparatorString** extends Object implementing DataComparator

This is a class that compares two records using strings. It is used to call the sortRecord method of the database.

**Fields inherited from the interface org.kwis.msp.db.DataComparator**

EQUIVALENT, FOLLOWS, PRECEDES

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

## Detailed Description of the Generator

- **DataComparatorString**

***public DataComparatorString(int offset)***

This is a class that compares two records using strings.

When calling the sortRecord method in a database, the class that implemented the DataComparator interface can pass on this class or DataComparatorInteger. sortRecord will then convert the byte array, starting with the offset that is passed on when this class is created into a string, and carry out the comparison.

This is required when calling sortRecord.

**Parameters**

offset                      Byte offset where the string to be compared starts in the record

## Detailed Description of the Method

- **compare**

***public int compare(byte[] data1, byte[] data2)***

This is a method (comparator) that compares records. In implementing this method, note that the byte arrays received as parameters should follow the format of the record data stored in the database.

**Parameters**

data1    Record data to be compared

data2    Record data to be compared

**Return Value**

DataComparator.EQUIVALENT when two records are equivalent in sequence;  
DataComparator.FOLLOWS when data2 is followed by data1 (i.e., data1 follows  
2); DataComparator.PRECEDES when data1 is followed by data2



**Class DataFilterInteger**

java.lang.Object

|

+---org.kwis.msp.db.**DataFilterInteger**

All implemented interfaces: DataFilter

public class **DataFilterInteger** extends Object implementing DataFilter

This class limits records that will be used in sorting. It is used when sorting integers.

Assuming the data stored in a specific offset is an integer, only the records wherein this value is in a certain range (specified in min and max) are included in the sorting in byte arrays that are record data.

This class is used to call the sortRecord method.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field**

- **offset**

*protected int offset*

Offset to be compared

- **min**

*protected int min*

Minimum value

- **max**

*protected int max*

Maximum value

**Detailed Description of the Generator**

- **DataFilterInteger**

*public DataFilterInteger(int offset, int min, int max) throws**IllegalArgumentException*

This generator limits the records that will be used in sorting. It is used when sorting integers. Assuming a specific position (offset) is an integer in record data, this generator indicates whether this record is within a certain range as expressed by max and min. All byte sequences follow big-endian. When sorting records, this value is used to sort only those records that belong to a certain range.

**Parameters**

offset    Offset to be compared  
min       Minimum value; Integer.MIN\_VALUE means below maximum  
max       Maximum value; Integer.MAX\_VALUE means above minimum

**Throws**

IllegalArgumentException    Issued when offset is a negative value, or min/max  
is not adequately addressed

**Detailed Description of the Method**

- **filter**

***public boolean filter(byte[] data)***

This is a method that limits records to be used in sorting and determines whether or not a certain record will be used in sorting. In implementing this method, note that the byte arrays received as parameters should follow the format of the record data stored in the database.

**Parameters**

data    Byte array representing data stored in a record

**Return Value**

True when the record is included in sorting; otherwise, False is returned

**Class DataBaseException**

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--org.kwis.msp.db.DataBaseException
  
```

public class **DataBaseException** extends Exception

Occurs in an exceptional situation related to a database

**Methods inherited from class java.lang.Throwable**

getMessage, printStackTrace, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **DataBaseException**

***public DataBaseException(String message)***

Creates a new DataBaseException instance together with a message

**Parameters**

message      Detailed message on an exception

- **DataBaseException**

***public DataBaseException()***

Creates a new DataBaseException instance

**Class DataBaseRecordException**

```

java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--org.kwis.msp.db.DataBaseRecordException

```

public class **DataBaseRecordException** extends Exception

Occurs in an exceptional situation related to a database

**Methods inherited from class java.lang.Throwable**

getMessage, printStackTrace, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **DataBaseRecordException**

***public DataBaseRecordException(String message)***

Creates a new DataBaseRecordException instance together with a message

**Parameters**

message      Detailed message on an exception

- **DataBaseRecordException**

***public DataBaseRecordException()***

Creates a new DataBaseRecord instance

### 3.1.4. High-Level User Interface

#### Interface ActionListener

public interface **ActionListener**

This is an interface that is called when a certain action occurs. It is a listener processing an event that occurs when the user presses a component, e.g., button.

#### Detailed Description of the Method

- **action**

***public void action(Component cmp, Object o)***

Called when an action occurs in a button or list

**Parameters**

- cmp    Component where an action occurred
- obj    Object argument entered when this interface is set

**Interface ChangeListener**

public interface **ChangeListener**

This is an interface that is called when the selection is changed. It is a listener processing an event that occurs when there is a change in the selection status in a list or choice group.

**Detailed Description of the Method**

- **changed**

***public void changed(Component cmp, Object obj)***

This method is called when a change event occurs in a component wherein ChangeListener is set. In case of a change in the status of CheckboxComponent or ListComponent, the change method of the registered listener is called.

**Parameters**

- cmp    Component wherein an event occurred
- obj    Object (extended parameter) set by setChangeListener

**Reference Item**

CheckboxComponent.setChangeListener(org.kwis.msp.lwc.ChangeListener,  
java.lang.Object),  
CheckboxGroup.setChangeListener(org.kwis.msp.lwc.ChangeListener,  
java.lang.Object),  
ListComponent.setChangeListener(org.kwis.msp.lwc.ChangeListener,  
java.lang.Object),  
ProgressComponent.setChangeListener(org.kwis.msp.lwc.ChangeListener,  
java.lang.Object)

**Interface CommandListener**

public interface **CommandListener**

An interface that indicates content selection or focus change of a command, i.e.,  
Select/Change

**Detailed Description of the Field**

- **FOCUS\_CHANGE**

*public static final int FOCUS\_CHANGE*

Constant when a command focus is changed

- **SELECT**

*public static final int SELECT*

Constant when a command is selected

**Detailed Description of the Method**

- **commandAction**

*public void commandAction(Command c, int type, Object obj)*

Called when command content is selected, or command focus is changed

**Parameters**

- |      |   |
|------|---|
| C    | Command receiving a selected command or a focus                           |
| type | SELECT when selecting a command; FOCUS_CHANGE when changing command focus |

**Interface EventListener**

public interface **EventListener**

An interface indicating events that occur in a component, such as key, show, focus, and pointer event

**Detailed Description of the Method**

- **eventNotify**

***public boolean eventNotify(int type, int arg1, int arg2, int arg3, Object obj)***

Called when events occur in a component, such as key, show, focus, and pointer

**Parameters**

- |      |   |
|------|---|
| c    | Command receiving a selected command or focus   |
| type | Represents the event type that occurred (Component.KEY_NOTIFY, Component.SHOW_NOTIFY, Component.FOCUS_NOTIFY, Component.POINTER_NOTIFY) |
| arg1 | Used variably depending on the value of type  |
| arg2 | Used variably depending on the value of type  |
| arg3 | Used variably depending on the value of type  |
| obj  | Object set by setEventListener  |



**Interface GrabKeyListener**

public interface **GrabKeyListener**

An interface that indicates the occurrence of a grabbed key event when Key Grab is set

**Detailed Description of the Method**

- **grabKeyNotify**

***public boolean grabKeyNotify(int type, int ch, Object obj)***

Called when an event occurs in a component, such as key, show, focus, or pointer

**Parameters**

- |      |   |
|------|---|
| c    | Command receiving a selected command or focus   |
| type | Represents the event type that occurred (EventQueue.KEY_RELEASED, EventQueue.KEY_RELEASED, EventQueue.KEY_REPEATED, EventQueue.KEY_TYPED) |
| chr  | Key value that occurred   |
| obj  | Object set by setEventListener  |

**Class AnnunciatorComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
    |
    +--org.kwis.msp.lwc.ContainerComponent
        |
        +--org.kwis.msp.lwc.ShellComponent
            |
            +--org.kwis.msp.lwc.AnnunciatorComponent

```

public class **AnnunciatorComponent** extends ShellComponent

This is a class that shows the strength of electric wave and the capacity of battery use to the user. When various internal values change, the content of the display also changes in this class.

**Fields inherited from class org.kwis.msp.lwc.ShellComponent**

cd, cmpCommand, cmpTitle, cmpWork, RESIZE\_MASK

**Fields inherited from class org.kwis.msp.lwc.ContainerComponent**

cmpFocus, cmps, insetBottom, insetLeft, insetRight, insetTop, ncomp, offsetX, offsetY, useFrame

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.ShellComponent**

addComponent, configure, controlInset, getCard, getCommand, getNextTraversalComponent, getPrevTraversalComponent, getTitle, getWorkComponent, getX, getY, grabKey, isShown, keyNotify, processEvent, removeAllComponents, repaint, serviceRepaints, setCommand, setGrabKeyListener, setTitle, setTitle, setWorkComponent, showNotify, ungrabKey

**Methods inherited from class org.kwis.msp.lwc.ContainerComponent**

getComponent, getIndexOf, getNumberOfComponent, paintFrame, removeComponent, repaint, scrollTo, setComponent, useFrame, validate

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, focusNotify, getBackground, getForeground, getHeight,

```
getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getXOnScreen,
getYOnScreen, hasFocus, invalidate, isValid, paintContent, pointerNotify, setBackground,
setEventListener, setFocus, setForeground, toString
```

#### Methods inherited from class java.lang.Object

```
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

### Detailed Description of the Generator

- **AnnunciatorComponent**

#### *public AnnunciatorComponent(boolean bTrans)*

This is a method that creates annunciator components. When setting bTrans to False, the content of the annunciator is shown on the upper part of the display, which cannot be used by the application program. On the other hand, when setting bTrans to True, the upper part of the display can be used by the application program. The content of the annunciator is displayed above the content drawn by the application program.

#### Parameters

bTrans Whether the annunciator will be displayed transparently or non-transparently

### Detailed Description of the Method

- **addComponent**

#### *public void addComponent(int idx, Component cmp)*

This method adds one child component indicated by cmp on the specified position. It is not provided in AnnunciatorComponent. Thus, IllegalStateException is issued every time this method is called.

#### Overrides

addComponent in class ShellComponent

#### Parameters

index Position where a component is to be added

cmp Component to be added

#### Throws

IllegalStateException Issued every time

- **removeComponent**

#### *public void removeComponent(Component cmp)*

This method removes a specified component. It is not provided in

AnnunciatorComponent; thus, `IllegalStateException` is issued every time this method is called.

**Overrides**

removeComponent in class ShellComponent

**Parameters**

cmp                      Component to be removed

**Throws**

`IllegalStateException`                      Issued every time

- **layout**

***public void layout()***

Determines the size and position of a child component

**Overrides**

layout in class ShellComponent

- **show**

***public void show()***

This method shows a component on the display. The position and size of a component are calculated using the `validate` method before the component is shown on the display. Since this component is directly added to the display, it should be shown first before the other components.

**Overrides**

show in class ShellComponent

- **hide**

***public void hide()***

This method hides a component. It is not provided in `AnnunciatorComponent`. Specifically, `AnnunciatorComponent` cannot be hidden once it is shown. Therefore, `IllegalStateException` is issued every time this method is called.

**Overrides**

hide in class ShellComponent

**Throws**

`IllegalStateException`                      Issued every time

- **paint**

***protected void paint(Graphics g)***

**Overrides**

paint in class ContainerComponent

This method paints a container component using graphics g. The container component even paints a child component using the paintContent method of the child component.

**Parameters**

g      Graphics object used to paint a component

**Class ButtonComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ButtonComponent

```

public class **ButtonComponent** extends Component

This is a button component. When the "select" key is pressed and released, this key calls ActionListener, which is registered in the button component. The button consists of one string and two images.

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **ButtonComponent**

***public ButtonComponent()***

Creates a button

- **ButtonComponent**

***public ButtonComponent(String str, Image img)***

This is a method that creates a button using specified images and string. Both the img and the str can be null.

**Parameters**

str      String for button

img      Image for button

### Detailed Description of the Method

- **setFont**

***public void setFont(Font ft)***

Sets the font for button

**Parameters**

ft      Font to be specified

- **getFont**

***public Font getFont()***

Returns font that is internally specified

**Return Value**

Specified font

- **setActionListener**

***public void setActionListener(ActionListener l, Object obj)***

This method registers ActionListener. When the button is pressed, ActionListener is called using the component and o as arguments. Currently registered ActionListener is replaced by a new ActionListener.

**Parameters**

l      ActionListener

obj      Argument to be passed on when this method is called

**Reference Item**

ActionListener.action(org.kwis.msp.lwc.Component, java.lang.Object)

- **keyNotify**

***public boolean keyNotify(int type, int chr)***

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

keyNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **setString**

***public void setString(String str)***

Specifies a string for the button

**Parameters**

str	String to be specified
-----	------------------------

- **getString**

***public String getString()***

Returns a string for the current button

**Return Value**

String for the current button

- **getImage**

***public Image getImage()***



Returns an image for the current button

**Return Value**

Image for the button

- **setImage**

***public void setImage(Image img)***

Specifies an image for the button

**Parameters**

img      Image to be specified; when img is null, the existing image is deleted

- **paintContent**

***public void paintContent(Graphics g)***

**Overrides**

paintContent in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g                  Graphics for painting

**Reference Item**

Graphics

- **layout**

- protected void layout()***

- Determines the size and position of a child component

- Overrides**

- layout in class Component

- **calcPreferredSize**

- protected void calcPreferredSize(int w)***

- Calculates the preferred size of the component

- Overrides**

- calcPreferredSize in class Component

**Class CheckboxComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
    |
    +--org.kwis.msp.lwc.LabelComponent
        |
        +--org.kwis.msp.lwc.CheckboxComponent
  
```

public class **CheckboxComponent** extends LabelComponent

CheckboxComponent is a class used to make a check button and a radio button that can be selected individually.

When a CheckboxComponent is created without the CheckboxGroup specification, it will operate as an independent checkbox. In case a CheckboxGroup is specified, however, CheckboxComponents bound as members of an identical CheckboxGroup will operate as a bound radio button.

The default value for CheckBoxes bound in an identical CheckboxGroup is an unselected state, except for the first component that is added. To change the default value, setState is used.

**Fields inherited from class org.kwis.msp.lwc.LabelComponent**

Layout, m\_f, m\_image, m\_str

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.LabelComponent**

CalcPreferredSize, getFont, getImage, getLabel, invalidate, setFont, setImage, setLabel, setLayout

**Methods inherited from class org.kwis.msp.lwc.Component**

canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, isShown, isValid, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

Equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **CheckboxComponent**

***public CheckboxComponent(String str, Image img)***

This method creates a new CheckboxComponent. This component will have a given string and several images. The default value for this CheckboxComponent is False.

**Parameters**

str      String for the Checkbox  
img      Image for the Checkbox

**Reference Item**

CheckboxComponent(String, Image, CheckboxGroup)  
CheckboxComponent(String, Image, boolean)  
CheckboxComponent(String, Image, CheckboxGroup, boolean)  
CheckboxGroup

- **CheckboxComponent**

***public CheckboxComponent(String str, Image img, CheckboxGroup cb)***

This method creates a new CheckboxComponent. This component will have a given string and several images, and it will be grouped as a member of CheckboxGroup. When CheckboxGroup is null, this component will become an independent Checkbox.

**Reference Item**

CheckboxComponent(String, Image)  
CheckboxComponent(String, Image, boolean)  
CheckboxComponent(String, Image, CheckboxGroup, boolean)  
CheckboxGroup

- **CheckboxComponent**

***public CheckboxComponent(String str, Image img, boolean bSet)***

This method creates a new CheckboxComponent. This component will have a given string and images, and the default state will be determined by bSet.

**Reference Item**

CheckboxComponent(String, Image)  
CheckboxComponent(String, Image, CheckboxGroup)  
CheckboxComponent(String, Image, CheckboxGroup, boolean)  
setState(boolean)  
CheckboxGroup

**Detailed Description of the Method**

- **setState**

***public void setState(boolean bState)***

This method changes the selection state of CheckboxComponent. In the case of CheckboxComponent in a group, the current selection state is managed by CheckboxGroup. When False is specified in setState, this input is ignored because it is impossible to determine which CheckboxComponent of the group should be selected. Therefore, no action is taken. When the selection state is changed using this method, ChangeListener is called.

**Reference Item**

getState()

- **getState**

***public boolean getState()***

Gets the selection state of CheckboxComponent

**Reference Item**

setState(boolean)

- **paintContent**

***public void paintContent(Graphics g)***

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class LabelComponent

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g          Graphics for painting

**Reference Item**

Graphics

- **keyNotify**

***public boolean keyNotify(int type, int key)***

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the key input value is ch.

**Overrides**

keyNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **setChangeListener**

***public void setChangeListener(ChangeListener listener, Object obj)***

This method registers ChangeListener in CheckboxComponent. When the selection state of CheckboxComponent is changed, the change method of the registered Listener is called.

**Parameters**

Listener	Listener that is called
obj	Object (extended parameter) to be passed on when Listener is called

**Reference Item**

CheckboxGroup.setChangeListener(org.kwis.msp.lwc.ChangeListener,  
java.lang.Object)

**Class CheckboxGroup**

java.lang.Object

|

+---org.kwis.msp.lwc.**CheckboxGroup**public class **CheckboxGroup** extends Object

CheckboxGroup binds several CheckboxComponents in a group to enable their operation as a grouped radio button.

CheckboxComponents registered as CheckboxGroup cannot have several ON states concurrently; only one CheckboxComponent can be ON concurrently. Therefore, when one Checkbox is ON, all other Checkboxes in the group are OFF. The default value of CheckboxComponent initially registered is ON.

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

- **CheckboxGroup**

***public CheckboxGroup()***

Creates a new CheckboxGroup

**Reference Item**

CheckboxComponent

**Detailed Description of the Method**

- **select**

***public void select(CheckboxComponent cb)***

Among CheckboxComponents bound in CheckboxGroup, only the state of the selected CheckboxComponent will be ON. The state of other Checkboxes in the group will be OFF.

**Parameters**

cb      CheckboxComponent to be selected

**Reference Item**

getSelectedCheckbox()



- **getSelectedCheckbox**

***public CheckboxComponent getSelectedCheckbox()***

Gets CheckboxComponent whose current state is ON from among Checkboxes registered in CheckboxGroup

**Return Value**

CheckboxComponent whose current state is ON

**Reference Item**

select(org.kwis.msp.lwc.CheckboxComponent)

- **setChangeListener**

***public void setChangeListener(ChangeListener listener, Object obj)***

This method registers ChangeListener in CheckboxGroup. In case of a change in the state of CheckboxComponent registered in CheckboxGroup, the change method of the registered Listener is called.

**Parameters**

listener            Listener that is called

obj                Object (extended parameter) to be received when Listener is called

**Reference Item**

CheckboxComponent.CheckboxComponent(String, Image, CheckboxGroup,  
boolean)

CheckboxComponent.CheckboxComponent(String, Image, CheckboxGroup)

CheckboxComponent.setChangeListener(org.kwis.msp.lwc.ChangeListener,  
java.lang.Object)

**Class ComboComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ComboComponent

```

public class **ComboComponent** extends Component

ComboComponent is a class providing the pop-up menu method.

It is divided into an area that shows a selected item from among pop-up menu items and an area that shows a list of pop-up menus. Entering the SELECT key on the display showing a selected item causes a pop-up menu list to be displayed. Selecting a certain item causes the pop-up menu list to close. A new selected item is then displayed.

ChangeListener can be registered in ComboComponent, which is used to monitor the change of selected items. When selecting a new item on the pop-up menu, the change method of ChangeListener is called.

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **ComboComponent**

***public ComboComponent()***

Creates a ComboComponent instance

**Detailed Description of the Method**

- **append**

***public int append(String str)***

Creates a new item of the pop-up menu with a given character data and appends it to the bottom of the pop-up menu list

**Parameters**

str      String to be appended

**Return Value**

Appended string index

**Throws**

IllegalArgumentException      Issued when str is null

- **insert**

***public int insert(int index, String str)***

This method creates a new item of the pop-up menu at a given index position using a given character data and inserts it in the pop-up menu list. In case of an index value that is smaller than 0 or larger than the number of elements appended to ListItemComponent, IndexOutOfBoundsException is issued.

**Parameters**

index    Index of an item to be inserted

**Return Value**

Index value inserted

**Throws**

IndexOutOfBoundsException      Issued when an invalid index value is  
specified

**Reference Item**

ListComponent.insert(int, java.lang.String, org.kwis.msp.lcd.ui.Image)

- **set**

***public void set(int index, String str)***

This method creates a pop-up menu item at a given index position using the given character data and specifies it anew. In case of an index value that is smaller than 0 or larger than the number of elements appended to ListItemComponent, IndexOutOfBoundsException is issued.

**Parameters**

index    Index of an item to be replaced

**Throws**

IndexOutOfBoundsException    Issued when an invalid index value is specified

- **delete**

***public void delete(int index)***

This method deletes an item at a given index position of the pop-up menu list. In case of an index value that is smaller than 0 or larger than the number of elements appended to ListItemComponent, IndexOutOfBoundsException is issued.

**Parameters**

index    Index of an item to be deleted

**Throws**

IndexOutOfBoundsException    Issued when an invalid index value is specified

- **getString**

***public String getString()***

Gets a string for the currently selected item

**Return Value**

String for the selected item; in case no item is selected, a null value is returned

- **getSize**

***public int getSize()***

Gets the number of the pop-up menu list items of ComboComponent

**Return Value**

Number of items contained

- **getSelectedIndex**

***public int getSelectedIndex()***

The following explanation of this method is copied from the Component class.

Gets the index for the selected item from among pop-up menu list items

**Return Value**

Returns -1 in the absence of an index for the selected item

- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. Otherwise, in case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **getPreferredWidth**

***public int getPreferredWidth()***

This method determines the preferred width of the component. The value returned by this method is referred to when Container determines the size of the component.

**Overrides**

getPreferredWidth in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Width of the component

- **paintContent**

***public void paintContent(Graphics g)***

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

g      Graphics for painting

**Reference Item**

Graphics

- **keyNotify**

***protected boolean keyNotify(int type, int key)***

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is ch.

**Overrides**

keyNotify in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **setChangeListener**

***public void setChangeListener(ChangeListener listener, Object obj)***

Sets ChangeListener to monitor the change state when a selected item from among the pop-up menu list items is changed in ComboComponent

**Parameters**

listener	ChangeListener that has been implemented; when this value is null, the registered Listener will be canceled
obj	Object passed on as an argument when the change event occurs, and the change method of ChangeListener is called

- **select**

***public void select(int index)***

Selects a given index item in the pop-up menu list

**Parameters**

index	Index of an item to be selected
-------	---------------------------------

**Throws**

IndexOutOfBoundsException Issued when an invalid index value is specified



## Class Command

```
java.lang.Object
|
+--org.kwis.msp.lwc.Command
```

public class **Command** extends Object

This is a class representing commands at the disposal of the user. It defines commands that can be used by the user on UI components. A command is expressed by strings and images registered in CommandBarComponent. The size of an image is 20x20 pixel.

### ● Reference Item

CommandBarComponent

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Detailed Description of the Generator

#### ● Command

***public Command(String str, Object obj)***

This method creates a command with a specified string. In this case, the image is null. The user can set any specific object as obj, which can be read using the getExtObject method.

#### Parameters

str     String  
obj     Extended object

#### ● Command

***public Command(String str, Image img, Object obj)***

This method creates a command with a specified string and several images. The user can set any specific object as obj, which can be read using the getExtObject method.

#### Parameters

str     String  
img     Image  
obj     Extended object

- **Command**

***public Command(String str, Image img, Image imgActive, Object obj)***

This method creates a command with an image when a specified string and several images are selected. The user can set any specific object as obj, which can be read using the getExtObject method.

**Parameters**

str	String
img	Image
imgActive	Image that is selected
obj	Extended object

- **Command**

***public Command(String str, String imgString, Object obj)***

This method creates a command using a specified string and several images read from a specified resource.

Images are read by calling Image.loadImage when the getNormalImage method is called. The selected image will become a specified image. The user can set any specific object as obj, which can be read using the getExtObject method.

**Parameters**

str	String
imgString	String representing the path name of the image resource
obj	Extended object

- **Command**

***public Command(String str, String imgString1, String imgString2, Object obj)***

This method creates a command using a specified string and several images read from a specified resource.

Images are read by calling Image.loadImage when the getNormalImage method is called. Specify the resource path name of a common image for imgString1 and the resource path name of the selected image for imgString2. The selected image will become a specified image. The user can set any specific object as obj, which can be read using the getExtObject method.

**Parameters**

str	String
imgString1	String representing the resource path name of the image resource
imgString2	String representing the resource path name of the image resource
obj	Extended object

**Detailed Description of the Method**

- **getString**

***public String getString()***

Returns the string that describes a command stored internally

**Return Value**

String representing a command

- **getExtObject**

***public Object getExtObject()***

Returns the object used to extend commands stored internally at the time of creation

**Return Value**

Extended object

- **getNormallImage**

***public Image getNormallImage()***

Returns the image when an image is specified and loads the resource from the string when a string of the resource with the image is specified using the Image.loadImage method

**Return Value**

Image

**Reference Item**

Image.loadImage(java.lang.String, org.kwis.msp.lcdui.ImageObserver)

- **getActiveImage**

***public Image getActiveImage()***

Returns the specified image and loads the resource from the string when a string of the resource with the image is specified using the Image.loadImage method

**Return Value**

Image

**Class CommandBarComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.CommandBarComponent

```

public class **CommandBarComponent** extends Component

This is a command component. This class constitutes more than one command registered in a bar, shows the commands at the disposal of the user on the display, and receives the command selected by the user. The default value for the active index is -1.

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **CommandBarComponent**

***public CommandBarComponent()***

Creates a command bar component

**Detailed Description of the Method**

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height for the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height for the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **getPreferredWidth**

***public int getPreferredWidth()***

The following explanation of this method is copied from the Component class.

This method determines the preferred width for the component. The value returned by this method is referred to when Container determines the size of the component.

**Overrides**

getPreferredWidth in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Width of the component

- **getSize**

***public int getSize()***

Gets the number of registered commands

**Return Value**

Number of registered commands

- **addCommand**

***public int addCommand(Command cmd)***

This method adds a command. When a command is initially added, the active index is -1. The command for the desired index should be activated using setActiveIndex.

**Parameters**

cmd    Command to be added

**Return Value**

Index for the added command; returns -1 in case of failure

**Reference Item**

Command, setActiveIndex(int)

- **removeCommand**

***public void removeCommand(Command cmd)***

Removes a specified command

**Parameters**

cmd    Command to be removed

**Reference Item**

Command

- **removeAll**

***public void removeAll()***

Removes all commands

**Reference Item**

Command

- **setActiveIndex**

***public void setActiveIndex(int index)***

Sets selected commands

**Parameters**

index   Index for the command to be selected

- **getActiveIndex**

***public int getActiveIndex()***

This method returns the index for a selected command. When there is no index selected, -1 is returned.

**Return Value**

Index for the selected command

- **getCommand**

***public Command getCommand(int index)***

This method returns a command corresponding to the index. When there is no command corresponding to the index, a null value is returned.

**Parameters**

index   Index for the command to be returned

**Return Value**

Specified command

- **setCommandListener**

***public void setCommandListener(CommandListener cl, Object obj)***

This method sets Command Listener. It is called when a command is selected, or the command focus has been changed.

**Parameters**

cl      CommandListener  
obj     Object that is passed on at commandAction

- **keyNotify**

***protected boolean keyNotify(int type, int chr)***

The following explanation of this method is copied from the Component class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

keyNotify in class Component  
Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

type            Type of key input, i.e., KEY\_PRESSED when a key is pressed;  
KEY\_RELEASED when a key is released; KEY\_REPEATED when a  
key is pressed and held; and KEY\_TYPED when a key is pressed  
once and released  
chr             Character of the key pressed; default values are numbers 0 to 9 and  
symbols \* and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component;  
otherwise, False is returned



- **pointerNotify**

***protected boolean pointerNotify(int type, int x, int y)***

The following explanation of this method is copied from the Component class.

This method is called when a pointer input is received. Since there is no pointing device in all the currently available handsets, however, this method is not called.

**Overrides**

pointerNotify in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

type    Type of the pointing device

x        X coordinate of the device

y        Y coordinate of the device

**Return Value**

False when the event passed on as an argument is processed by this component

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

g        Graphics for painting

**Reference Item**

Graphics

**Class Component**

```
java.lang.Object
|
+--org.kwis.msp.lwc.Component
```

Directly known subclasses:

ButtonComponent, ComboComponent, CommandBarComponent, ContainerComponent, DateFieldComponent, ImageComponent, LabelComponent, ProgressComponent, ScrollbarComponent, TextComponent, TickerComponent

```
public abstract class Component extends Object
```

This is a class that is shown on the primary display.

Having position and size, this class receives the user's input and carries out an appropriate action.

All UI components shown on the display should inherit this class for implementation.

This class has position and width on the parent component of a child class inheriting component class, background color, and characteristics of a component (whether or not it can be inputted, input focus, etc.).

A component class should always have a parent class. A component that does not need a parent should become ShellComponent. In other words, a component can be shown on the display only when there is more than one ShellComponent on the display. Once addComponent is implemented, a component cannot add more components to other parent components.

The width and area for all components are determined by the program. Nonetheless, there are instances when the size of a component is determined by a parent component. For example, in the case of LabelComponent, a component higher than FormComponent, the size of the component varies depending on the length of an internal string.

When there is a need to recalculate the size of a component, e.g., when the user changes the content of a component in the middle of implementation, the invalidate method is called. The validate() method is then called when the component is shown on the display, or the paintContent method is called. The appropriate size of a component, including that of a child component, can be calculated using this method.

A component has a method that calculates and returns its appropriate size depending on the content of the component. This method is used to determine the size of a child component in the layout method of the parent ContainerComponent. In addition, there is a method that gets an appropriate height when a specific width is given for a component that is capable of formatting (Label, TextField, TextArea).

The determination of size for a component is done by the parent component rather than by itself. The total size for the parent component is determined by the parent component of such parent component. The top-most component of all UI components should always be ShellComponent.

A component is responsible for processing an event. When True is returned by the CanHandleInput method of the component, such component can be given an input focus by the setFocus method of ContainerComponent. In this case, the keyNotify method can be called. In addition, methods such as showNotify, focusNotify, and pointerNotify can be called. In particular, when a certain content has to be painted, the paintContent method is called.

When an event is processed by itself, the keyNotify method or pointerNotify method returns True. As a result, the key event is not transmitted to the parent component. When False is returned, however, the key event is transmitted to the parent component.

All events that occurred are disclosed to a specified EventListener through the setEventListener method. When True is returned by EventListener, the events are no longer processed. When False is returned, however, events are normally processed.

When implementing the paintContent method, the content of the graphics is received with a starting point and a clipping area of the graphics changed to fit the position and size of the component.

Changing the content of this clipping using the setClip method or re-initializing it through the reset method may pose certain problems. The content of the component may be printed in an unexpected place, or it may appear on the display belatedly.

#### Rule on Layout Combination

Layouts provided by the component include LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_HCENTER, LAYOUT\_TOP, LAYOUT\_BOTTOM, and LAYOUT\_VCENTER. For the following layouts, however, IllegalArgumentException is issued:

LAYOUT\_LEFT|LAYOUT\_RIGHT

LAYOUT\_TOP|LAYOUT\_BOTTOM

<b>Methods inherited from class java.lang.Object</b>
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

<b>Detailed Description of the Field</b>
--

- **LAYOUT\_LEFT**

***public static final int LAYOUT\_LEFT***

This is the left alignment value of the component. The position of images and character data is aligned to the left of the component area. A value of "1" is specified.

- **LAYOUT\_RIGHT**

***public static final int LAYOUT\_RIGHT***

This is the right alignment value of the component. The position of images and character data is aligned to the right of the component area. A value of "2" is specified.

- **LAYOUT\_HCENTER**

***public static final int LAYOUT\_HCENTER***

This is the horizontal center alignment value of the component. The position of images and character data is aligned to the center of the component horizon. A value of "4" is specified.

- **LAYOUT\_TOP**

***public static final int LAYOUT\_TOP***

This is the top alignment value of the component. The position of images and character data is aligned to the top of the component area. A value of "8" is specified.

- **LAYOUT\_BOTTOM**

***public static final int LAYOUT\_BOTTOM***

This is the bottom alignment value of the component. The position of images and character data is aligned to the bottom of the component area. A value of "16" is specified.

- **LAYOUT\_VCENTER**

***public static final int LAYOUT\_VCENTER***

This is the vertical center alignment value of the component. The position of images and character data is aligned to the center of the component vertical area. A value of "32" is specified.

- **x**

***protected int x***

X axis pixel position from the parent component

- **y**

***protected int y***

Y axis pixel position from the parent component

- **w**

***protected int w***

Pixel position of the component width

- **h**

***protected int h***

Pixel position of the component height

- **parent**

***protected ContainerComponent parent***

Parent component

- **bg**

***protected int bg***

Background color; a value of -1 indicates a transparent color

- **fg**

***protected int fg***

Foreground color; the default value is set differently depending on the component

- **evtListener**

***protected EventListener evtListener***

- **evtListenerObj**

***protected Object evtListenerObj***

- **prefW**

***protected int prefW***

- **prefH**

***protected int prefH***

- **POS\_MASK**

***public static final int POS\_MASK***

Constant indicating that the position can be moved

**Reference Item**

configure(int, int, int, int, int)

- **SIZE\_MASK**

***public static final int SIZE\_MASK***

Constant indicating that size change is possible

**Reference Item**

configure(int, int, int, int, int)

- **mask**

***protected int mask***

- **VALID\_MASK**

***protected static final int VALID\_MASK***

- **HAS\_FOCUS\_MASK**

***protected static final int HAS\_FOCUS\_MASK***

- **INPUT\_MASK**

***protected static final int INPUT\_MASK***

- **PREFER\_SIZE\_MASK**

***protected static final int PREFER\_SIZE\_MASK***

- **FOCUS\_NOTIFY**

***public static final int FOCUS\_NOTIFY***

Constant indicating that a focus has arrived (specified as 1)

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **SHOW\_NOTIFY**

***public static final int SHOW\_NOTIFY***

Constant indicating Show or Hide (specified as 2)

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **KEY\_NOTIFY**

***public static final int KEY\_NOTIFY***

Constant indicating the creation of a key-related event (specified as 3)

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **POINTER\_NOTIFY**

***public static final int POINTER\_NOTIFY***

Constant indicating the creation of a pointer-related event (specified as 4)

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **KEY\_PRESSED**

***public static final int KEY\_PRESSED***

Event type when a is pressed

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **KEY\_RELEASED**

***public static final int KEY\_RELEASED***

Event type when a key is released

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **KEY\_REPEATED**

***public static final int KEY\_REPEATED***

Event type when a key is pressed repeatedly

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **KEY\_TYPED**

***public static final int KEY\_TYPED***

Event type when a key is typed

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **POINT\_PRESSED**

***public static final int POINT\_PRESSED***

Event type when a pointer device is pressed

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **POINT\_RELEASED**

***public static final int POINT\_RELEASED***

Event type when a pointer device is released

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

- **POINT\_DRAGGED**

***public static final int POINT\_DRAGGED***

Event type when a pointer device is moved while pressed

**Reference Item**

setEventListener(org.kwis.msp.lwc.EventListener, java.lang.Object)

### Detailed Description of the Generator

- **Component**

***protected Component()***

Used by an inherited component

### Detailed Description of the Method

- **getWidth**

***public int getWidth()***

Returns the component width

**Return Value**

Width of the component

- **getHeight**

***public int getHeight()***

Returns the component height

**Return Value**

Height of the component

- **canHandleInput**

***public boolean canHandleInput()***

This method indicates whether or not the component can receive an input. A return value of true means that the current component can receive a focus or can handle an input.

By implementing the setFocus method of ContainerComponent, this method can receive all key inputs from the user. Changing the mask value in class generator enables determining whether or not an input is possible in the component.



**Return Value**

Whether or not receiving an input is possible

- **hasFocus**

***public boolean hasFocus()***

This method indicates whether or not the component has an input focus. When the component has input focus, and the user presses the button, the keyNotify method of this component is called. The key inputted by the user is received as an argument of such method.

**Return Value**

Whether or not the component has a focus

- **configure**

***public void configure(int x, int y, int w, int h, int mask)***

This method changes the position and the size of a component depending on the mask. A logical AND operation is carried out for the mask value and POS\_MASK. This method changes a value of POS\_MASK to positions x and y in the parent component. Similarly, a logical AND operation is carried out for the mask value and SIZE\_MASK. If the value is SIZE\_MASK, this method changes the size of the component to (w, h). In other words, the size and the position of the component can be changed at the same time. This method calls the repaint method for the changed area, causing the area to be painted through the paintContent method. The size of the component is determined by the layout method of the parent component.

**Parameters**

x	X coordinate of the component on the parent component
y	Y coordinate of the component on the parent component
w	Width of the component
h	Height of the component
mask	POS_MASK   SIZE_MASK can be received; when POS_MASK is received, x and y values become valid values (when SIZE_MASK is received, w and h values become valid values)

- **getX**

***public int getX()***

Returns the x coordinate of the component on the parent component

**Return Value**

X coordinate

- **getY**

***public int getY()***

Returns the y coordinate of the component on the parent component

**Return Value**

Y coordinate

- **calcPreferredSize**

***protected void calcPreferredSize(int w)***

Calculates the preferred size of the component

- **getPreferredHeight**

***public int getPreferredHeight()***

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **getPreferredWidth**

***public int getPreferredWidth()***

This method determines the preferred width of the component. The value returned by this method is referred to when Container determines the size of the component.

**Return Value**

Width of the component

- **setBackground**

***public void setBackground(int bg)***

This method sets the background color. In case of a background color of -1, the background color is not painted since it should be transparent. The value for the color is 0x00RRGGBB.

**Parameters**

bg      Background color of the component

**Reference Item**

getBackground()

- **setForeground**

***public void setForeground(int fg)***

This method sets the foreground color. It is used in a component that uses foreground color. The default value is set differently depending on the component.

**Parameters**

fg      Foreground color of the component

**Reference Item**

setBackground(int bg)

- **getBackground**

***public int getBackground()***

Returns the background color

**Return Value**

Background color of the component

**Reference Item**

setBackground(int)

- **getForeground**

***public int getForeground()***

Returns the foreground color

**Return Value**

Foreground color of the component

**Reference Item**

setForeground(int)

- **paintContent**

***public void paintContent(Graphics g)***

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Parameters**

g Graphics for painting

**Reference Item**

Graphics

- **isValid**

***protected boolean isValid()***

This method indicates whether or not the component has valid coordinates and size. In case the size has to be changed considering the changed content of the component, False is returned. Otherwise, True is returned.

**Return Value**

Whether or not the component is valid

- **invalidate**

***public void invalidate()***

Indicates that the component does not have valid coordinates or size



- **validate**

***public void validate()***

This method causes a component to have valid coordinates and size. It is called in paintContent and showNotify. The layout method is also called to enable the component to have valid coordinates. The layout does not determine its own size; instead, it determines the size of a child component based on its own size.

- **layout**

***protected void layout()***

Determines the size and position of a child component

- **setFocus**

***public void setFocus()***

This method sets a focus to enable a key event to be transmitted. It causes the user's key input to be passed on to its own component.

**Reference Item**

focusNotify(boolean)

- **focusNotify**

***public void focusNotify(boolean b)***

This method is called upon receiving a focus. To show whether or not the component has a focus, this method calls the repaint method to make it redraw itself.

**Parameters**

b        True when the component has a focus; otherwise, False is returned

- **keyNotify**

***protected boolean keyNotify(int type, int chr)***

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Parameters**

type        Type of key input, i.e., KEY\_PRESSED when a key is pressed;  
KEY\_RELEASED when a key is released; KEY\_REPEATED when a  
key is pressed and held; and KEY\_TYPED when a key is pressed

once and released

chr      Character of the key pressed; default values are numbers 0 to 9 and symbols \* and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **pointerNotify**

***protected boolean pointerNotify(int type, int x, int y)***

This method is called when a pointer input is received. Since there is no pointing device in all the currently available handsets, however, this method is not called.

**Parameters**

type    Type of the pointing device

x        X coordinate of the device

y        Y coordinate of the device

**Return Value**

False is returned when the event passed on as an argument is processed by this component; otherwise, True is returned

- **showNotify**

***protected void showNotify(boolean bShow)***

This method is called when the content of the display is shown. It is called using addComponent or removeComponent, or when the top-most parent component (ShellComponent) is shown on the display.

**Parameters**

bShow   Whether or not the component is shown

- **repaint**

***public void repaint()***

This method is called when there is a need to override the display content. In this case, the entire component is overridden. Finally, this method calls the repaint of the card, and the called repaint method automatically undergoes a process of calling the paintContent method of the component after a certain period of time.

- **isShown**

***public boolean isShown()***

This method indicates whether or not the current component is shown. True is returned when the current component is shown on the display. Otherwise, False is returned.

**Return Value**

Whether or not the component is shown

- **processEvent**

***protected boolean processEvent(int type, int subtype, int param1, int param2)***

Processes an event

**Parameters**

type	Event type
subtype	Sub-event type subject to an event type
param1	Additional argument
param2	Additional argument

- **repaint**

***public void repaint(int x, int y, int w, int h)***

This method is called when there is a need to update the display content. Finally, this method calls the repaint of the card, and the called repaint method automatically undergoes a process of calling the paint() method of the component after a certain period of time.

**Parameters**

x	X coordinate of the area to be overridden
y	Y coordinate of the area to be overridden
w	Width of the area to be overridden
h	Height of the area to be overridden

- **serviceRepaints**

***public void serviceRepaints()***

This method prints the repainted content immediately on the display. Instead of calling paint later through repaint, this method calls the paint method directly.



- **toString**

***public String toString()***

Converts an object into a string, representing the component as a string

**Overrides**

toString in class Object

**Return Value**

String representing the component

- **getXOnScreen**

***public int getXOnScreen()***

Gets the x coordinate of the component on the upper left corner of the display

**Return Value**

X coordinate on the display

- **getYOnScreen**

***public int getYOnScreen()***

Gets the y coordinate of the component on the upper left corner of the display

**Return Value**

Y coordinate on the display

- **getCard**

***public Card getCard()***

This method returns the card that is connected to the current component. A parent component like ShellComponent has an internal card. This method returns the card of the top-most parent component of the component. In the absence of a parent component, however, a null value can be returned.

**Return Value**

Card that is connected to the component

**Reference Item**

Card

- **setEventListener**

***public void setEventListener(EventListener listener, Object obj)***

This method registers EventListener. Specifically, this method transmits an event to a specified EventListener. When True is returned, the event is not processed. Otherwise, when False is returned, the event is processed.

**Parameters**

listener Event listener

obj      Parameter when the event listener is called

**Reference Item**

EventListener

**Class ContainerComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ContainerComponent

```

Directly known subclasses:  
FormComponent, ShellComponent

public abstract class **ContainerComponent** extends Component

This component can method as the parent component of another component. It determines the position and size of a child component and manages its focus. A component can be registered as a child component using the addComponent method and can be removed using the removeComponent method. A component has a parent component. The top-most component is ShellComponent, which is shown on the display using the show method. ContainerComponent determines the size and location of child components through the layout method. There is an inset inside a container. Child components appear only inside this inset; they are not printed outside of the inset. To enable a specific component to receive a key input, the setFocus method should be called.

- **Reference Item**

ShellComponent, Component

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getPreferredSize, getPreferredWidth, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, paintContent, pointerNotify, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString

**Methods inherited from class java.lang.Object**

Equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **cmps**

***protected Component[] cmps***

Internal child components

- **ncomp**

***protected int ncomp***

Number of internal child components

- **cmpFocus**

***protected Component cmpFocus***

Component with a focus

- **offsetX**

***protected int offsetX***

Position where internal components are scrolled

- **offsetY**

***protected int offsetY***

Position where internal components are scrolled

- **insetTop**

***protected short insetTop***

Top height of inset

- **insetBottom**

***protected short insetBottom***

Bottom height of inset

- **insetLeft**

***protected short insetLeft***

Left width of inset

- **insetRight**

***protected short insetRight***

Right width of inset

- **useFrame**

***protected boolean useFrame***

Whether or not a frame is used

## Detailed Description of the Generator

- **ContainerComponent**

***protected ContainerComponent()***

Used by the inherited component

**Detailed Description of the Method**● **addComponent*****public void addComponent(int index, Component cmp)***

This method adds a component indicated by cmp on the specified position.

When cmp is null, NullPointerException is issued. On the other hand, when cmp has another parent component, IllegalArgumentException is issued.

In case of an index value that is smaller than 0 or larger than the number of added components, IndexOutOfBoundsException is issued.

**Parameters**

index    Position where a component is to be added  
 cmp     Component to be added

**Throws**

IllegalArgumentException	Issued when cmp has another parent component
IndexOutOfBoundsException	Issued when the index is out of bounds
NullPointerException	Issued when cmp is null

● **addComponent*****public int addComponent(Component cmp)***

Adds a child component at the very top

**Parameters**

cmp     Child component to be added

**Throws**

IllegalArgumentException	Issued when cmp has another parent component
NullPointerException	Issued when cmp is null

● **setComponent*****public void setComponent(int index, Component cmp)***

This method replaces a component of the specified index with a given component. When cmp is null, NullPointerException is issued. On the other hand, when cmp has another parent component, IllegalArgumentException is issued.

In case of an index value that is smaller than 0 or larger than the number of added components, IndexOutOfBoundsException is issued.

**Parameters**

index    Position where the component is to be replaced  
cmp      Component to be replaced

**Throws**

IllegalArgumentException    Issued when cmp has another parent component  
NullPointerException      Issued when cmp is null  
IndexOutOfBoundsException   Issued when the index is out of bounds

- **removeComponent**

***public void removeComponent(int index)***

This method removes an item of the specified order. The component whose number matches the number of index is removed.

In case of an index value that is smaller than 0 or larger than the number of added components, IndexOutOfBoundsException is issued.

**Parameters**

index    Component to be removed

**Throws**

IndexOutOfBoundsException   Issued when the index is out of bounds

- **removeComponent**

***public void removeComponent(Component cmp)***

This method removes a specified component. When the specified component is not registered as a child component, IllegalArgumentException is issued.

**Parameters**

cmp      Component to be removed

- **removeAllComponents**

***public void removeAllComponents()***

Removes all components

- **getComponent**

***public Component getComponent(int i)***

This method brings a component in a specific sequence of stack. When the index is out of bounds, a null value is returned.

**Parameters**

i      Sequence of stack to enable the component to be brought

**Return Value**

Component

- **getIndexOf**

***public int getIndexOf(Component cmp)***

Brings the stack sequence of the component

**Parameters**

cmp      Component for bringing the sequence

**Return Value**

Stack sequence

- **validate**

***public void validate()***

The following explanation of this method is copied from the Component class.

This method causes a component to have valid coordinates and size. It is called in paintContent and showNotify. The layout method is also called to enable the component to have valid coordinates. The layout does not determine its own size; instead, it determines the size of a child component based on its own size.

**Overrides**

validate in class Component

- **getNextTraversalComponent**

***protected Component getNextTraversalComponent()***

This method returns the next traversal component. The component to be returned will be one of the child components of the present component. In the absence of a traversal component, a null value is returned.

**Return Value**

Next traversal component

- **getPrevTraversalComponent**

***protected Component getPrevTraversalComponent()***

This method returns the previous traversal component. The component to be returned

will be one of the child components of the present component. In the absence of a traversal component, a null value is returned.

#### Return Value

Previous traversal component

- **keyNotify**

***protected boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the Component class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

#### Overrides

keyNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

#### Parameters

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

#### Return Value

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **processEvent**

***protected boolean processEvent(int type, int subtype, int param1, int param2)***

The following explanation of this method is copied from the Component class.

Processes an event

#### Overrides

processEvent in class Component

Copied from class: org.kwis.msp.lwc.Component





**Parameters**

type	Event type
subtype	Sub-event type subject to an event type
param1	Additional argument
param2	Additional argument

● **paint*****protected void paint(Graphics g)***

This method paints a ContainerComponent with graphics g. In this case, the ContainerComponent even paints a child component using the paintContent method of a child component.

**Parameters**

g	Graphics object used to paint the component
---	---

● **scrollTo*****protected boolean scrollTo(int dx, int dy)***

This method scrolls the display to a specific position. to come up with offset values represented by dx and dy. When the values cannot be processed, scrollTo of the parent component is called.

**Parameters**

dx	Distance for scrolling toward the x axis
dy	Distance for scrolling toward the y axis

**Return Value**

True in case of normal scrolling; otherwise, False is returned

● **repaint*****public void repaint(int x, int y, int w, int h)***

The following explanation of this method is copied from the Component class.

This method is called when there is a need to override the display content. It calls the repaint of the card, and the called repaint method automatically undergoes a process of calling the paint() method of the component after a certain period of time.

**Overrides**

repaint in class Component  
Copied from class: org.kwis.msp.lwc.Component

**Parameters**

x	X coordinate of the area to be overridden
y	Y coordinate of the area to be overridden
w	Width of the area to be overridden
h	Height of the area to be overridden

- **repaint**

***public void repaint()***

The following explanation of this method is copied from the Component class.

This method is called when there is a need to override the display content. The entire component is overridden. Finally, this method calls the repaint of the card, and the called repaint method automatically undergoes a process of calling the paintContent method of the component after a certain period of time.

**Overrides**

repaint in class Component

- **useFrame**

***public void useFrame(boolean useFrame)***

This method specifies whether or not ContainerComponent will print the frame on the display. When True is returned, setInset() is called to specify the frame value. Otherwise, when False is specified, initInset() is called to initialize the frame value to 0.

- **controllInset**

***protected void controllInset(boolean flag)***

This method controls the value for the frame thickness to be used by ContainerComponent. When True is returned, the value for frame thickness specified by each ContainerComponent is applied. Otherwise, when False is specified, the currently specified value for frame thickness is initialized to 0.

- **paintFrame**

***protected void paintFrame(Graphics g)***

Called to paint a frame when an argument of useFrame is called with True

**Parameters**

g	Graphics object for painting
---	------------------------------

- **getNumberOfComponent**

***public int getNumberOfComponent()***

Gets the number of registered components

**Return Value**

Number of currently registered components

**Class DateFieldComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.DateFieldComponent

```

public class **DateFieldComponent** extends Component

DateFieldComponent is used to display the fields that show the date and time on the display and to modify these values.

Regardless of the specified mode when creating DateFieldComponent, this class will have data initialized to the present time and date using the default TimeZone and Data set in the system. These values can be obtained or modified through getDate and setDate.

The date and time of DateFieldComponent will be printed on the display according to the specified mode, and the values can be modified using the arrow keys.

DateFieldComponent provides three types of modes that can be used to specify or modify time and date:

MODE\_TIME prints the field showing the time on the display. This can be used to modify time.

MODE\_DATE prints the field showing the date on the display. This can be used to modify date.

MODE\_TIME\_DATE prints the field showing the date and time on the display. This can be used to modify both time and date.

- **Reference Item**

Date, Calendar, TimeZone

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, pointerNotify, processEvent, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **MODE\_TIME**

***public static final int MODE\_TIME***

This is the time display mode. In this mode, DateFieldComponent prints the field showing the time on the display, which can be used to modify the time.

The value for MODE\_TIME is set to 0.

- **MODE\_DATE**

***public static final int MODE\_DATE***

This is the date display mode. In this mode, DateFieldComponent prints the field showing the date on the display, which can be used to modify the date.

The value for MODE\_DATE is set to 1.

- **MODE\_TIME\_DATE**

***public static final int MODE\_TIME\_DATE***

This is the date and time display mode. In this mode, DateFieldComponent prints the field showing the time and date on the display, which can be used to modify the time and date.

The value for MODE\_TIME\_DATE is set to 2.

**Detailed Description of the Generator**

- **DateFieldComponent**

***public DateFieldComponent(int mode)***

This method generates an instance of DateFieldComponent using default TimeZone and Date, which are initialized to the current time and date.

Fields to be used are determined based on a mode value given. When the mode value is MODE\_TIME, the field that shows the time is printed on the display, which can be used to modify the time. On the other hand, when MODE\_DATE is specified as the

mode value, the field that shows the date is printed on the display, which can be used to modify the date. `MODE_TIME_DATE` prints the field that shows both the time and date on the display, which can be used to modify the time and date.

When values other than said modes are specified, `IllegalArgumentException` is issued.

#### Parameters

mode Mode value for `DateFieldComponent` to be generated

#### Throws

`IllegalArgumentException` Issued when values other than the three modes declared (`MODE_TIME`, `MODE_DATE`, `MODE_TIME_DATE`) are specified

#### Reference Item

`MODE_TIME`, `MODE_DATE`, `MODE_TIME_DATE`, `TimeZone`, `Date`

### Detailed Description of the Method

- **getDate**

***public Date getDate()***

This method gets the date object with date information currently set using `DateFieldComponent`. Regardless of the mode specified, this value returns the currently set Date object. The default date when `DateFieldComponent` is created has a time value that is calculated based on 1970/1/00:00:00 GMT in millisecond.

#### Reference Item

`setDate(Date dt)`, `Date`

- **getMode**

***public int getMode()***

Gets the mode set in `DateFieldComponent`

#### Reference Item

`setMode(int mode)`

- **getTimeZone**

***public TimeZone getTimeZone()***

This method gets the TimeZone set in DateFieldComponent. The basic TimeZone set is the default TimeZone of the system, which is created through TimeZone.getTimeZone().

**Return Value**

TimeZone set in DateFieldComponent

**Reference Item**

setTimeZone(TimeZone tz), TimeZone

- **showNotify**

***protected void showNotify(boolean bShow)***

The following explanation of this method is copied from the Component class.

This method is called when the display content is shown. This method is called using addComponent or removeComponent, or when the top-most component (ShellComponent) of the component is shown on the display by show.

**Overrides**

showNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

bShow Whether or not the component is shown

- **setDate**

***public void setDate(Date dt)***

This method sets a date in DateFieldComponent. When the argument value is null, NullPointerException is issued. The default date has a time value that is calculated based on 1970/1/00:00:00 GMT in millisecond.

**Parameters**

dt Date object to be set in DateFieldComponent

**Throws**

NullPointerException issued when date is null



**Reference Item**

getDate(), Date

- **setMode**

***public void setMode(int mode)***

This method sets a mode in DateFieldComponent. The values available for setting include MODE\_TIME, MODE\_DATE, and MODE\_TIME\_DATE. When a value other than these is set, IllegalArgumentException is issued.

**Parameters**

mode Mode to be set in DateFieldComponent

**Throws**

IllegalArgumentException Issued when a value other than what is specified is given as an argument

**Reference Item**

getMode(), MODE\_TIME, MODE\_DATE, MODE\_TIME\_DATE

- **setTimeZone**

***public void setTimeZone(TimeZone tz)***

This method sets TimeZone in DateFieldComponent. The basic TimeZone set is the default TimeZone of the system, which is created through TimeZone.getTimeZone().

**Parameters**

tz TimeZone to be set in DateFieldComponent

**Throws**

NullPointerException Issued when TimeZone is null

**Reference Item**

getTimeZone(), TimeZone

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

**The following explanation of this method is copied from the Component class.**

This method determines the preferred height of the component. The height of the component is returned when the component has a specified, limited width. If it is a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

● **getPreferredHeight**

***public int getPreferredHeight()***

**The following explanation of this method is copied from the Component class.**

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

● **getPreferredWidth**

***public int getPreferredWidth()***

**The following explanation of this method is copied from the Component class.**

This method determines the preferred height of the component. The value returned by this method is referred to when Container determines the size of the component.

**Overrides**

getPreferredWidth in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Width of the component

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g Graphics for painting

**Reference Item**

Graphics

- **keyNotify**

***public boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the Component class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

keyNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **getStringValue**

***public String getStringValue(int mode)***

This method gets the value for data or time in string form based on the mode value given as an argument. The format to be returned depending on the mode is as follows:

MODE_DATE	Mon, 10 Dec 2001
MODE_TIME	03:28 (am)
MODE_TIME_DATE	Mon, 10 Dec 2001 03:28 (am)

**Parameters**

mode	DateFieldComponent mode
------	-------------------------

**Return Value**

Value for date/time that fits the mode in string form

**Throws**

IllegalArgumentException	Issued when an invalid mode value is specified
--------------------------	--

### Class DialogComponent

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
    |
    +--org.kwis.msp.lwc.ContainerComponent
        |
        +--org.kwis.msp.lwc.ShellComponent
            |
            +--org.kwis.msp.lwc.DialogComponent
  
```

public class **DialogComponent** extends ShellComponent

DialogComponent is a component that is created to support various forms of dialog boxes.

Basically, DialogComponent consists of a title area, a data area, and a button area. When there is a title in the dialog box, the title area is an area where the title is displayed. The data area is where various forms of components added by the user are displayed, whereas the button area is where various buttons in use by type are displayed. Except for the character data and components in each area, the position of each area cannot be changed.

In the data area of DialogComponent, only one component can be added. Therefore, when creating a dialog with several components, it is necessary to use ContainerComponent to add them to the data area.

Basically, DialogComponent supports three types: TYPE\_OK type that provides the confirmation method, TYPE\_OK\_CANCEL type that provides confirmation and cancel methods, and TYPE\_NONE type that provides a method displaying data on the screen for a certain period of time.

In the case of TYPE\_NONE, the time data displayed on the screen is basically three seconds, and this time value can be specified by the user using the setTimeout(int) method. The TIMEOUT\_INFINITE value specified as timeout is changed to the TYPE\_OK type.

When an invalid type is specified, or a value other than TYPE\_NONE, TYPE\_OK, or TYPE\_OK\_CANCEL is specified, IllegalArgumentException is issued.

#### Fields inherited from class org.kwis.msp.lwc.ShellComponent

cd, cmpCommand, cmpTitle, cmpWork, RESIZE\_MASK

#### Fields inherited from class org.kwis.msp.lwc.ContainerComponent

cmpFocus, cmps, insetBottom, insetLeft, insetRight, insetTop, ncomp, offsetX, offsetY, useFrame

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.ShellComponent**

addComponent, configure, controlInset, getCard, getCommand, getNextTraversalComponent, getPrevTraversalComponent, getTitle, getWorkComponent, getX, getY, grabKey, hide, isShown, keyNotify, processEvent, removeAllComponents, removeComponent, repaint, serviceRepaints, setCommand, setGrabKeyListener, setTitle, setTitle, setWorkComponent, showNotify, ungrabKey

**Methods inherited from class org.kwis.msp.lwc.ContainerComponent**

getComponent, getIndexOf, getNumberOfComponent, paint, removeComponent, repaint, scrollTo, setComponent, useFrame, validate

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, focusNotify, getBackground, getForeground, getHeight, getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getXOnScreen, getYOnScreen, hasFocus, invalidate, isValid, paintContent, pointerNotify, setBackground, setEventListener, setFocus, setForeground, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **TYPE\_NONE**

***public static final int TYPE\_NONE***

This is a dialog type without buttons. The default display time is 3 seconds, with a specified value of "0."

- **TYPE\_OK**

***public static final int TYPE\_OK***

A dialog type with only the OK button; a value of "1" is specified

- **TYPE\_OK\_CANCEL**

***public static final int TYPE\_OK\_CANCEL***

A dialog type with OK and CANCEL buttons; a value of "2" is specified

- **DLG\_TIMEOUT**

***public static final int DLG\_TIMEOUT***

Value to be returned on doModal; this field represents termination or TIMEOUT (a value of "10" is specified)

- **DLG\_OK**

***public static final int DLG\_OK***

Value to be returned on doModal; this field represents the OK button (a value of "11" is specified)

- **DLG\_CANCEL**

***public static final int DLG\_CANCEL***

Value to be returned on doModal; this field represents the CANCEL button (a value of "12" is specified)

- **OK\_BUTTON**

***public static final int OK\_BUTTON***

OK button type; a value of "20" is specified

- **CANCEL\_BUTTON**

***public static final int CANCEL\_BUTTON***

CANCEL button type; a value of "21" is specified

- **TIMEOUT\_INFINITE**

***public static final int TIMEOUT\_INFINITE***

Represents infinity among the timeout values; a value of "-1" is specified

- **actionState**

***protected int actionState***

This is a field that remembers which action (event) has occurred. When a button action has occurred, this field specifies the value for the button; on the other hand, when a timeout action has occurred, this field specifies the timeout value. When an action occurs, and the present value is changed from -2 to a different value, the current DialogComponent의 doModal() is terminated, and DialogComponent is removed from the display.

### Detailed Description of the Generator

- **DialogComponent**

***public DialogComponent(int type)***

This method generates a new instance of DialogComponent with a given type instead of data component and title.

The width and height of DialogComponent are automatically specified according to the values of the width and height of a new component added to DialogComponent. When values for the width and height of the data exceed the values for the width and height of the display, the values for the width and height of the display are used.

When an invalid data value or a value other than TYPE\_NONE, TYPE\_OK, or TYPE\_OK\_CANCEL is specified, IllegalArgumentException is issued.

**Parameters**

type    Type of the dialog box

**Throws**

IllegalArgumentException	Issued when a type other than TYPE_NONE, TYPE_OK, or TYPE_OK_CANCEL is specified
IllegalArgumentException	Issued when cmp has another parent component

**Reference Item**

TYPE\_NONE, TYPE\_OK, TYPE\_OK\_CANCEL



- **DialogComponent**

***public DialogComponent(Component cmp, String title, int type)***

This method generates a new instance of DialogComponent.

The width and height of DialogComponent are specified according to the values of the width and height of a new component added to DialogComponent. When values for the width and height of the data exceed values for the width and height of the display, values for the width and height of the display are used.

When an invalid data value or a value other than TYPE\_NONE, TYPE\_OK, or TYPE\_OK\_CANCEL is specified, IllegalArgumentException is issued.

Both the component to be specified for the title area and the component to be specified for the data area can be null.

**Parameters**

cmp     Component with the content of the dialog box; may also be a null value  
 title   Title of the dialog box; may also be a null value  
 type    Type of the dialog box

**Throws**

IllegalArgumentException     Issued when a type other than TYPE\_NONE,  
    TYPE\_OK, or TYPE\_OK\_CANCEL is specified  
 IllegalArgumentException     Issued when cmp has another parent component

**Reference Item**

TYPE\_NONE, TYPE\_OK, TYPE\_OK\_CANCEL

- **DialogComponent**

***public DialogComponent(Component cmp, String ttl, int type, int x, int y, int w, int h)***

This method generates a new stance of DialogComponent.

The position, area, and height of DialogComponent can be specified. When the specified position value is smaller than 0, or the size value is 0 or less, IllegalArgumentException is issued.

When an invalid type or a type other than TYPE\_NONE, TYPE\_OK, or TYPE\_OK\_CANCEL is specified, IllegalArgumentException is issued.

Both the component to be specified for the title area and the component to be specified for the data area can be null.

**Parameters**

cmp	Component with the content of the dialog box; may also be a null value
title	Title of the dialog box; may also be a null value
type	Type of the dialog box (TYPE_NONE, TYPE_OK, or TYPE_OK_CANCEL)
x	X coordinate of the dialog box
y	Y coordinate of the dialog box
w	Value for the width of the dialog box
h	Value for the height of the dialog box

**Throws**

IllegalArgumentException	Issued when a type other than TYPE_NONE, TYPE_OK, or TYPE_OK_CANCEL is specified
IllegalArgumentException	Issued when values for x and y are smaller than 0, or values for w and h are 0 or less
IllegalArgumentException	Issued when cmp has another parent component

**Reference Item**

TYPE\_NONE, TYPE\_OK, TYPE\_OK\_CANCEL

**Detailed Description of the Method**

- **setButtonString**

***public void setButtonString(int buttonType, String buttonStr)***

This method sets the characters for a button. When the type of DialogComponent is not TYPE\_NONE, characters used for a button can be changed depending on the type. Input the button type and new characters for the button to be changed. The type of button that can be specified is either OK\_BUTTON or CANCEL\_BUTTON. Otherwise, when a button type other than these two is specified, IllegalArgumentException is issued. When the type is TYPE\_NONE, no action is taken.

**Parameters**

buttonType	Button type to be specified (either OK_BUTTON or CANCEL_BUTTON)
buttonStr	New string for the button

**Throws**

IllegalArgumentException	Issued when an invalid button type is specified
--------------------------	---

**Reference Item**

OK\_BUTTON, CANCEL\_BUTTON

- **setType**

***public void setType(int type)***

This method sets the type of DialogComponent. When the type is changed, the currently specified timeout value is changed to the default timeout value by type. The default timeout value for TYPE\_NONE is 3 seconds, whereas the default timeout value for other types is TIMEOUT\_INFINITE.

When an invalid type or a type other than TYPE\_NONE, TYPE\_OK, or TYPE\_OK\_CANCEL is specified, IllegalArgumentException is issued.

TYPE\_OK and TYPE\_OK\_CANCEL are shown on the display until there is user input. When removing the type from the display after a certain period of time, specify the time value using setTimeout(int).

**Parameters**

tp      Type of DialogComponent

**Throws**

IllegalArgumentException      Issued when an invalid type value is specified

**Reference Item**

setTimeout(int timeout), TYPE\_NONE, TYPE\_OK, TYPE\_OK\_CANCEL

- **setTimeout**

***public void setTimeout(int timeout)***

This method sets the timeout value for DialogComponent. This value is applicable only to the current type. When the type is changed, the timeout value is also changed to the default value by type (refer to setType(int)).

The unit for the timeout value is millisecond. In case the current type is TYPE\_NONE, the current type is changed to TYPE\_OK when a value for TYPE\_NONE is specified.

Once set, the timeout value remains valid until it is reset, except in case of a type change.

**Parameters**

timeout Time displayed on screen

**Reference Item**

getTimeout(), setType(int type)

- **getTimeout**

***public int getTimeout()***

This method gets the currently set timeout value. The unit for the timeout value is millisecond.

**Return Value**

Current timeout value

**Reference Item**

setTimeout(int timeout)

- **doModal**

***public int doModal()***

This method causes DialogComponent to be shown on the display. When calling this method, a dialog box is shown on the display. In case of an action through a button or a timeout, the dialog box is removed from the display. The value for the action is returned.

**Return Value**

Value for the action DLG\_OK, DLG\_CANCEL, DLG\_TIMEOUT

- **show**

***public void show()***

The following explanation of this method is copied from the ShellComponent class.

This method shows a component on the display. Before showing the component on the display, this method calculates its position and size using the validate method.

**Overrides**

show in class ShellComponent

- **getActionState**

***public int getActionState()***

Gets the last action that occurred in DialogComponent

**Return Value**

Last action that occurred

- **layout**

***public void layout()***

The following explanation of this method is copied from the Component class.

Determines the position and size of a child component

**Overrides**

layout in class ShellComponent



- **paintFrame**

***protected void paintFrame(Graphics g)***

The following explanation of this method is copied from the **ContainerComponent** class.

Called for painting a frame when calling an argument of useFrame with True

**Overrides**

paintFrame in class ContainerComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Parameters**

g      Graphics object for painting

**Class FormComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ContainerComponent
|
+--org.kwis.msp.lwc.FormComponent

```

Directly known subclasses:  
ListComponent

public class **FormComponent** extends ContainerComponent

This is a component that arrays various components in a row to constitute a display. It expands ContainerComponent to contain various components as child components in order to constitute a display. This component also manages the layout and scrolling of components.

The focus movement of child components allows only upward and downward movements. The focus of a child component can be moved using the UP/DOWN key. When there are many internal components, a scrollbar is automatically generated.

**Fields inherited from class org.kwis.msp.lwc.ContainerComponent**

cmpFocus, cmps, insetBottom, insetLeft, insetRight, insetTop, ncomp, offsetX, offsetY, useFrame

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.ContainerComponent**

addComponent, addComponent, controllInset, getComponent, getIndexOf, getNumberOfComponent, paintFrame, processEvent, removeAllComponents, removeComponent, removeComponent, repaint, repaint, setComponent, useFrame, validate

**Methods inherited from class org.kwis.msp.lwc.Component**

canHandleInput, configure, getBackground, getCard, getForeground, getHeight, getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, paintContent, pointerNotify, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **FormComponent**

***public FormComponent()***

Generates FormComponent

- **FormComponent**

***public FormComponent(boolean bVertical)***

This method generates a vertical or horizontal FormComponent. When bVertical is True, components are aligned vertically; otherwise, when bVertical is False, components are aligned horizontally.

**Parameters**

bVertical      Whether components are aligned horizontally or vertically

**Detailed Description of the Method**

- **setPacked**

***public void setPacked(boolean b)***

Specifies whether or not the width of the internal components of a form should be fitted to the width of the form

**Parameters**

b      True when fitted to the width; otherwise, False is returned



**getPacked*****public boolean getPacked()***

Indicates whether or not the width of a child component should be fitted

**Return Value**

Whether or not the width of a child component should be fitted to the width of the form

- **setGap**

***public void setGap(int gap)***

Determines the gap between components

**Parameters**

gap     Gap between components

- **getGap**

***public int getGap()***

Returns the gap between components

**Return Value**

Gap between components

- **focusNotify**

***public void focusNotify(boolean b)***

**The following explanation of this method is copied from the Component class.**

This method is called when receiving a focus. To show whether or not a component has a focus, the repaint method is called to repaint itself.

**Overrides**

focusNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

b     True when it has a focus; otherwise, False is returned

- **keyNotify**

***protected boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the Component class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is ch.

**Overrides**

keyNotify in class ContainerComponent

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **calcPreferredSize**

***protected void calcPreferredSize(int cw)***

The following explanation of this method is copied from the Component class.

Calculates the preferred size for the component

**Overrides**

calcPreferredSize in class Component

- **layout**

***public void layout()***

The following explanation of this method is copied from the Component class.

Determines the size and position of the child component

**Overrides**

layout in class Component

- **paint**

***public void paint(Graphics g)***

This method paints a container component using graphics g. In this case, the container component even paints a child component using the paintContent method of the child component.

**Overrides**

paint in class ContainerComponent

**Parameters**

g Graphics object for painting

- **getNextTraversalComponent**

***protected Component getNextTraversalComponent()***

The following explanation of this method is copied from the ContainerComponent class.

This method returns the next component that can have a focus. The component to be returned becomes one of the child components of the current component. When there is no component that can have a focus, a null value is returned.

**Overrides**

getNextTraversalComponent in class ContainerComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Return Value**

Next component that can have a focus

- **getPrevTraversalComponent**

***protected Component getPrevTraversalComponent()***

The following explanation of this method is copied from the ContainerComponent class.

This method returns the previous component that can have a focus. The component to be returned becomes one of the child components of the current component. When there is no component that can have a focus, a null value is returned.

**Overrides**

getPrevTraversalComponent in class ContainerComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Return Value**

Previous component that can have a focus

- **scrollTo**

***protected boolean scrollTo(int dx, int dy)***

The following explanation of this method is copied from the ContainerComponent class.

This method scrolls the display to a specific position to enable it to have offset values represented by dx and dy. When the values cannot be processed, scrollTo of the parent component is called.

**Overrides**

scrollTo in class ContainerComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Parameters**

dx      Distance for scrolling toward the x axis

dy      Distance for scrolling toward the y axis

**Return Value**

True when scrolling is normal; otherwise, False is returned

**Class ImageComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ImageComponent

```

```

public class ImageComponent extends Component

```

ImageComponent is a class that prints image data on the display in a specified layout.

ImageComponent can specify the layout using setLayout(int). Layouts provided by ImageComponent include Component.LAYOUT\_LEFT, Component.LAYOUT\_RIGHT, Component.LAYOUT\_HCENTER, Component.LAYOUT\_TOP, Component.LAYOUT\_BOTTOM, and Component.LAYOUT\_HCENTER. Each layout can be combined by referring to the Rule of Layout Combination. LAYOUT\_LEFT|LAYOUT\_TOP is the default layout set at the creation of ImageComponent.

- **Reference Item**

Image

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, keyNotify, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **imgStr**

*protected String imgStr*

Image resource

- **img**

*protected Image img*

Image

**Detailed Description of the Generator**

- **ImageComponent**

*public ImageComponent()*

This method generates a new instance of ImageComponent. Image data is initialized to a null value.

- **ImageComponent**

*public ImageComponent(Image img)*

This method generates a new instance of ImageComponent. This method can assign a null value to img.

**Parameters**

img Image to be used as default value; may also be a null value

- **ImageComponent**

*public ImageComponent(String str)*

This method generates a new instance of ImageComponent. An image is created using the image resource given at the time of generation.

**Parameters**

str Image resource to be used as a default or null value

**Detailed Description of the Method**

- **setImage**

*public void setImage(Image img)*

This method sets an image to ImageComponent. Any set image will be reset using a new image given. After the change, it will be automatically repainted. When img is a null

value, the set image is removed.

**Parameters**

img      Image to be set

- **getImage**

***public Image getImage()***

Gets the image set in ImageComponent

**Return Value**

Set image

- **setImage**

***public void setImage(String str)***

This method creates and sets an image using the image resource path name given in ImageComponent. When the string is a null value, the existing image is removed. When the path name is not valid, however, IllegalArgumentException is issued.

**Parameters**

str      Image resource to be set

**Throws**

IllegalArgumentException      Issued when the image resource path name is null  
or invalid

- **setLayout**

***public void setLayout(int layout)***

This method sets the layout for an image. The default layout is Component.LAYOUT\_LEFT. Layouts used in ImageComponent include Component.LAYOUT\_LEFT, Component.LAYOUT\_RIGHT, Component.LAYOUT\_TOP, Component.LAYOUT\_BOTTOM, Component.LAYOUT\_HCENTER, and Component.LAYOUT\_VCENTER. These layout values are referred to by the component.

**Parameters**

type      Value for the layout type

**Reference Item**

Component.LAYOUT\_LEFT, Component.LAYOUT\_RIGHT,  
Component.LAYOUT\_HCENTER, Component.LAYOUT\_TOP,  
Component.LAYOUT\_BOTTOM, Component.LAYOUT\_VCENTER





- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **getPreferredWidth**

***public int getPreferredWidth()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The value returned by this method is referred to when Container determines the size of the component.

**Overrides**

getPreferredWidth in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Width of the component

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g      Graphics for painting

**Reference Item**

Graphics

- **showNotify**

***protected void showNotify(boolean bShow)***

The following explanation of this method is copied from the Component class.

This method is called by addComponent or removeComponent, or when the top-most component (ShellComponent) of the component is shown on the display by show.

**Overrides**

showNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

bShow Whether or not the component is shown

- **play**

***public void play()***

Starts animation when the image is an animation image

- **stop**

***public void stop()***

Stops animation when the image is an animation image

## Class LabelComponent

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.LabelComponent

```

Directly known subclasses:

CheckboxComponent, ListItemComponent

public class **LabelComponent** extends Component

This is a component that shows strings. Before showing strings and images to the user, they are formatted for display. LabelComponent can set the layout using `setLayout(int)`. Layouts used in LabelComponent refer to the Rule on Layout Combination provided by the component. `LAYOUT_LEFT` is the default layout at the creation of LabelComponent. Strings or images can be null.

### Fields inherited from class org.kwis.msp.lwc.Component

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

### Methods inherited from class org.kwis.msp.lwc.Component

canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, isShown, isValid, keyNotify, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **layout**

***protected int layout***

This is the layout of LabelComponent. The default layout is Component.LAYOUT\_LEFT.

- **m\_ft**

***protected Font m\_ft***

Font used in character data

- **m\_str**

***protected String m\_str***

Character data

- **m\_image**

***protected Image m\_image***

Image data

**Detailed Description of the Generator**

- **LabelComponent**

***public LabelComponent()***

This method generates LabelComponent. String and image are set to null. LAYOUT\_LEFT is the default layout.

- **LabelComponent**

***public LabelComponent(String str)***

This method generates LabelComponent with the given string. Image data is set to null, although string data can also be null. LAYOUT\_LEFT is the default layout.

**Parameters**

str      String to be shown by LabelComponent; may also be a null value

- **LabelComponent**

***public LabelComponent(String str, Image img)***

This method generates a LabelComponent with the given string and image data. Both the string and the image data can be null. LAYOUT\_LEFT is the default layout.

**Parameters**

str      String to be shown by LabelComponent; may also be a null value

img      Image; may also be a null value

- **LabelComponent**

***public LabelComponent(String str, String imgString)***

This method generates LabelComponent with the given string and the image read from the specified resource. Both the string and the image data can be null. LAYOUT\_LEFT is the default layout.

**Parameters**

str	String to be shown by LabelComponent; may also be a null value
imgString	String representing the path name for the image resource; may also be a null value

### Detailed Description of the Method

- **invalidate**

***public void invalidate()***

The following explanation of this method is copied from the Component class.

Indicates that the component does not have valid coordinates and size

**Overrides**

invalidate in class Component

- **setLabel**

***public void setLabel(String str)***

This method sets an internal string with a given string value. The current string data can be set to null.

**Parameters**

str	String to be changed; may also be a null value
-----	--

- **setImage**

***public void setImage(Image img)***

This method sets an internal image with a given image. The current image data can be set to null.

**Parameters**

img	String to be changed; may also be a null value
-----	--

- **getLabel**

***public String getLabel()***

Gets an internal string

**Return Value**

Internal output String

- **getImage**

***public Image getImage()***

Gets an internal image

**Return Value**

Internal output Image

- **setFont**

***public void setFont(Font font)***

Sets the font (the font is set using Font.getDefaultFont() as a default value)

**Parameters**

font     User font

- **getFont**

***public Font getFont()***

Gets the font (the font is set using Font.getDefaultFont() as a default value)

**Return Value**

Set font

- **calcPreferredSize**

***protected void calcPreferredSize(int cw)***

The following explanation of this method is copied from the Component class.

Calculates the preferred size of the component

**Overrides**

calcPreferredSize in class Component

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g      Graphics for painting

**Reference Item**

Graphics

- **setLayout**

***public void setLayout(int layout)***

This method sets the layout for a label. The layout used in LabelComponent refers to the Rule on Layout Combination provided by the component. When a value other than the defined layouts is specified, IllegalArgumentException is issued.

**Parameters**

type    Layout

**Throws**

IllegalArgumentException      Issued when a value other than the defined layouts is specified in the component

**Reference Item**

Component.LAYOUT\_LEFT, Component.LAYOUT\_RIGHT,  
Component.LAYOUT\_HCENTER, Component.LAYOUT\_TOP,  
Component.LAYOUT\_VCENTER, Component.LAYOUT\_BOTTOM



### Class ListComponent

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ContainerComponent
|
+--org.kwis.msp.lwc.FormComponent
|
+--org.kwis.msp.lwc.ListComponent

```

public class **ListComponent** extends FormComponent

ListComponent is a class that is implemented by inheriting FormComponent. Unlike ContainerComponent, this class can add only ListItemComponent, and items are printed on the display according to its sequence.

There are three types of ListItemComponent\*: SELECT\_IMPLICIT, which has the same state as when an item that currently receives a focus is selected; SELECT\_MULTIPLE, which allows the selection of multiple items, and; SELECT\_EXCLUSIVE, which allows the selection of only one item. For SELECT\_MULTIPLE and SELECT\_EXCLUSIVE, the item with a focus and the selected item are not always identical.

When an additional ListItemComponent is selected using the SELECT key input, or a number key input for the relevant position is received in the number column, the selection action can be detected through the registration of ActionListener. Any selected item changed by arrow key input can be detected through the registration of ChangedListener.

When ListComponent is used, a number image is displayed as a default value. Control for the number image is implemented by #controlNumberImage(boolean showImage). When True is returned, a number image is printed on the display; otherwise, when False is specified, the number image is not printed.

#### ● Reference Item

ListItemComponent, ActionListener, ChangeListener

#### Fields inherited from class org.kwis.msp.lwc.ContainerComponent

cmpFocus, cmps, insetBottom, insetLeft, insetRight, insetTop, ncomp, offsetX, offsetY, useFrame

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.FormComponent**

calcPreferredSize, focusNotify, getGap, getPacked, layout, scrollTo, setGap, setPacked

**Methods inherited from class org.kwis.msp.lwc.ContainerComponent**

controllInset, getComponent, getIndexOf, getNumberOfComponent, paintFrame, processEvent, removeAllComponents, removeComponent, removeComponent, repaint, repaint, useFrame, validate

**Methods inherited from class org.kwis.msp.lwc.Component**

canHandleInput, configure, getBackground, getCard, getForeground, getHeight, getPreferredSize, getPreferredSize, getPreferredSize, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, paintContent, pointerNotify, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **SELECT\_IMPLICIT**

***public static final int SELECT\_IMPLICIT***

When ListComponent has a focus, the currently focused item and the selected item are identical. Therefore, only one item is selected for this type.

- **SELECT\_EXCLUSIVE**

***public static final int SELECT\_EXCLUSIVE***

For this type, only one item can be selected. The currently focused item and the selected item are not always identical.

- **SELECT\_MULTIPLE**

***public static final int SELECT\_MULTIPLE***

For this type, multiple items can be selected. The currently focused item is not always selected.

#### Detailed Description of the Generator

- **ListComponent**

***public ListComponent(int type)***

This method generates an instance of ListComponent, which does not contain items and whose size is 0 in a given type. When a type other than SELECT\_IMPLICIT, SELECT\_EXCLUSIVE, and SELECT\_MULTIPLE supported by ListComponent is specified, IllegalArgumentException is issued.

**Parameters**

type     Type of ListComponent

**Throws**

IllegalArgumentException     Issued when an invalid type is specified

**Reference Item**

ListComponent()

#### Detailed Description of the Method

- **append**

***public int append(String str, Image img)***

This method creates ListItemComponent using image data and character data given to ListComponent and appends them. Both character data and image data can be null.

**Parameters**

str     Character data of ListItemComponent; may also be a null value

img     Image data of ListItemComponent; may also be a null value

**Return Value**

Index value of an item after the item is appended

- **addComponent**

***public void addComponent(int index, Component cmp)***

Adds a component to the index position

When the component to be added is not ListItemComponent, IllegalArgumentException is issued. On the other hand, when cmp is null, NullPointerException is issued. Similarly, when cmp has another parent component, IllegalArgumentException is issued.

In case of an index value that is smaller than 0 or larger than the number of items added to ListComponent, IndexOutOfBoundsException is issued.

**Overrides**

addComponent in class ContainerComponent

**Parameters**

index    Position where a new component is to be added

cmp      Component to be added

**Throws**

IllegalArgumentException	Issued when the component to be added is not ListItemComponent
IllegalArgumentException	Issued when cmp has another parent component
IndexOutOfBoundsException	Issued when an invalid index is specified
NullPointerException	Issued when cmp is null

**Reference Item**

addComponent(Component cmp), setComponent(int index, Component cmp)

- **addComponent**

***public int addComponent(Component cmp)***

This method adds a child component to the very top.

When the component to be added is not ListItemComponent, IllegalArgumentException is issued. On the other hand, when cmp is null, NullPointerException is issued. Similarly, when cmp has another parent component, IllegalArgumentException is issued.

**Overrides**

addComponent in class ContainerComponent

**Parameters**

cmp    Component to be added

**Return Value**

Position index being located

**Throws**

IllegalArgumentException	Issued when the component to be added is not ListItemComponent
IllegalArgumentException	Issued when cmp has another parent component already
NullPointerException	Issued when cmp is null

**Reference Item**

addComponent(int index, Component cmp), setComponent(int index, Component cmp)

- **setComponent**

***public void setComponent(int index, Component cmp)***

This method replaces a child component of the specified index with a given component.

In case of an index value that is smaller than 0 or larger than the number of components added to ListComponent, IndexOutOfBoundsException is issued.

When the component to be specified is not ListItemComponent, IllegalArgumentException is issued. On the other hand, when cmp is null, NullPointerException is issued. Similarly, when cmp has another parent component, IllegalArgumentException is issued.

**Overrides**

setComponent in class ContainerComponent

**Parameters**

index    Position of the component to be specified

cmp    Component to be specified

### Throws

IllegalArgumentException	Issued when the component to be added is not ListItemComponent
IllegalArgumentException	Issued when cmp has another parent component
IndexOutOfBoundsException	Issued when an invalid index is specified
NullPointerException	Issued when cmp is null

### Reference Item

addComponent(int index, Component cmp), addComponent(Component cmp)

### ● insert

#### ***public int insert(int index, String str, Image img)***

This method creates ListItemComponent using character data and image data given to a relevant position and appends them.

In case of an index value that is smaller than 0 or larger than the number of elements added to ListComponent, IndexOutOfBoundsException is issued. When the component to be specified is not ListItemComponent, IllegalArgumentException is issued.

### Parameters

index	Index value to be added
str	Character data to be added
img	Image data added together with character data

### Return Value

Position index value added

### Throws

IndexOutOfBoundsException	Issued when an invalid index is specified
IllegalArgumentException	Issued when the component to be specified is not ListItemComponent
IllegalArgumentException	Issued when cmp has another parent component
IndexOutOfBoundsException	Issued in case of an index that is smaller than 0 or larger than the number of components added

### ● set

#### ***public void set(int index, String str, Image img)***

This method creates ListItemComponent using character data and image data given to a relevant position and sets them again. Both image data and character data can be null.

In case of an index value that is smaller than 0 or larger than the number of elements added to ListItemComponent, IndexOutOfBoundsException is issued.

**Parameters**

index    Index value  
str      Character data to be specified  
img      Image data added together with character data

**Throws**

IndexOutOfBoundsException    Issued when an invalid index is specified

- **getString**

***public String getString(int index)***

This method gets the string data of ListItemComponent in a given position. In case of an index value that is smaller than 0 or larger than the number of elements added to ListItemComponent, IndexOutOfBoundsException is issued.

**Parameters**

index    Index of the preferred element

**Return Value**

Element string corresponding to a given index

**Throws**

IndexOutOfBoundsException    Issued when an invalid index is specified

- **getImage**

***public Image getImage(int index)***

This method gets the image data of ListItemComponent in a given position. In the absence of the image, a null value is returned. In case of an index value that is smaller than 0 or larger than the number of elements added to ListItemComponent, IndexOutOfBoundsException is issued.

**Parameters**

index    Index of the preferred element

**Return Value**

Element image corresponding to a given index

**Throws**

IndexOutOfBoundsException    Issued when an invalid index is specified





- **getSize**

***public int getSize()***

Indicates the number of elements added to ListComponent

**Return Value**

Number of elements contained in ListComponent

- **isSelected**

***public boolean isSelected(int index)***

This method indicates whether or not the element of a given index is currently selected. When the index is out of index bounds of ListItemComponent registered in ListComponent, i.e., a value that is smaller than 0 or larger than the total number of ListItemComponents (1), IndexOutOfBoundsException is issued.

**Return Value**

True when selected; otherwise, False is returned

**Throws**

IndexOutOfBoundsException Issued when an invalid index is specified (a value smaller than 0 or larger than the total number of ListItemComponents (1))

- **getSelectedIndexes**

***public int[] getSelectedIndexes()***

Gets the index of currently selected elements from among ListComponent elements (when there is no selected element, a null value is returned)

**Return Value**

Index of currently selected elements (when there is no selected element, a null value is returned)

- **getSelectedIndex**

***public int getSelectedIndex()***

Gets the index of currently selected elements from among ListComponent elements (when there is no selected element, "-1" is returned)

**Return Value**

Index of currently selected elements (when there is no selected element, "-1" is returned)

- **setActionListener**

***public void setActionListener(ActionListener l, Object o)***

This method registers ActionListener in ListComponent. When one of ListItemComponents added to ListComponent is selected by the EventQueue.FIRE key input, ActionListener.action(Component cmp, Object o) of the registered ActionListener is executed.

**Parameters**

- l        ActionListener to be registered in ListComponent
- o        Object to be registered in ListComponent together with ActionListener

- **setChangeListener**

***public void setChangeListener(ChangeListener l, Object o)***

This method registers ChangeListener in ListComponent. When the selection status of ListItemComponents added to ListComponent is changed, ChangeListener.changed(Component cmp, Object o) of the registered ChangeListener is executed.

**Parameters**

- l        ChangeListener to be registered in ListComponent
- o        Object to be registered in ListComponent together with ChangeListener

- **keyNotify**

***protected boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the Component class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

- keyNotify in class FormComponent
- Copied from class: org.kwis.msp.lwc.Component

**Parameters**

- type        Type of key input, i.e., KEY\_PRESSED when a key is pressed; KEY\_RELEASED when a key is released; KEY\_REPEATED when a key is pressed and held; and KEY\_TYPED when a key is pressed once and released
- chr        Character of the key pressed; default values are numbers 0 to 9 and

symbols \* and # (other characters are also possible)

### Return Value

True when the keys passed on as arguments are handled by this component;  
otherwise, False is returned

- **select**

#### ***public void select(int index)***

This method selects the element in a given position. In case of an index value that is smaller than "-1" or larger than the total number of elements, `IndexOutOfBoundsException` is issued.

When "-1" is specified, there will be no selected element. On the other hand, when the type of the current `ListComponent` is `SELECT_EXCLUSIVE`, the first item will be selected.

### Parameters

index                      Position value of the element to be selected

### Throws

`IndexOutOfBoundsException` Issued when an invalid index is specified  
(`index < -1 || index > (total number of ListItems)`)

- **select**

#### ***public void select(ListItemComponent cmpp)***

This method changes the selection status of a given element.

When the type of the present `ListComponent` is `SELECT_IMPLICIT` or `SELECT_MULTIPLE`, a given element is specified as selected. When the type is `SELECT_MULTIPLE`, and the current element status is selected, it is specified as not selected. Otherwise, if the current element status is unselected, it is specified as selected.

### Parameters

cmpp    Element to be selected

- **controlNumber**

#### ***public void controlNumber(boolean showImage)***

This method specifies whether or not number key control is to be used. When True is specified, the number is printed on the screen (1~9 and 0), and selection of `ListItem` by a number key is enabled. Otherwise, when False is returned, the number is not printed on the display, and selection by a number key is disabled. The default state is True, and the

number is printed on the screen. When False is returned, however, selection by number key is disabled.

**Parameters**

useImage      Whether or not the number image is to be used (True or False)

**Reference Item**

isControlNumber()

- **isControlNumber**

***public boolean isControlNumber()***

Returns the number key control status

**Return Value**

Number control status

**Reference Item**

controlNumber(boolean)

- **paint**

***public void paint(Graphics g)***

**Overrides**

paint in class FormComponent

This method paints ContainerComponent using graphics g. ContainerComponent even paints a child component using the paintContent method of the child component.

**Parameters**

g      Graphics for painting

- **getNextTraversalComponent**

***protected Component getNextTraversalComponent()***

The following explanation of this method is copied from the ContainerComponent class.

This method returns the next traversal component. The component to be returned will be one of the child components of the current component. When there is no traversal component, however, a null value is returned.

**Overrides**

getNextTraversalComponent in class FormComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Return Value**

Next traversal component

- **getPrevTraversalComponent**

***protected Component getPrevTraversalComponent()***

The following explanation of this method is copied from the **ContainerComponent** class.

This method returns the previous traversal component. The component to be returned will be one of the child components of the current component. When there is no traversal component, however, a null value is returned.

**Overrides**

getPrevTraversalComponent in class FormComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Return Value**

Previous traversal component

**Class ListItemComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
    |
    +--org.kwis.msp.lwc.LabelComponent
        |
        +--org.kwis.msp.lwc.ListItemComponent
  
```

public class **ListItemComponent** extends LabelComponent

This is ListItemComponent added to ListComponent for use. It is implemented by inheriting LabelComponent, whose basic method is similar to that of LabelComponent. Since this component has INPUT\_MASK, however, it can receive focus and input.

- **Reference Item**

ListComponent

**Fields inherited from class org.kwis.msp.lwc.LabelComponent**

layout, m\_f, m\_image, m\_str

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.LabelComponent**

calcPreferredSize, getFont, getImage, getLabel, paintContent, setFont, setImage, setLabel, setLayout

**Methods inherited from class org.kwis.msp.lwc.Component**

canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getPreferredHeight, getPreferredHeight, getPreferredWidth, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, keyNotify, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **ListItemComponent**

***public ListItemComponent(String str)***

This method generates an instance of ListItemComponent using a given string. Image data is specified to null, although string data can also be null. LAYOUT\_LEFT is the default layout.

**Parameters**

str      String to be shown by ListItemComponent; may also be a null value

- **ListItemComponent**

***public ListItemComponent(String str, Image img)***

Generates new ListItemComponent using a given character data and image data

**Parameters**

str      Character data of ListItem; may also be a null value

img      Image data of ListItem; may also be a null value

- **ListItemComponent**

***public ListItemComponent(String str, String imgString)***

This method generates an instance of ListItemComponent using a given string and image data read from the specified resource. Both the string data and image data can be null. LAYOUT\_LEFT is the default layout. When the image path name is specified improperly, the image data becomes null.

**Parameters**

str                      String to be shown by LabelComponent; may also be a null value

imgString              String representing the path name of the image  
resource; may also be a null value



**Detailed Description of the Method**

- **setState**

***public void setState(boolean bState)***

Specifies the selection status of ListItemComponent

**Parameters**

bState True when selected; otherwise, False is returned

- **getState**

***public boolean getState()***

Gets the current selection status

**Return Value**

True when selected; otherwise, False is returned

**Class ProgressComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ProgressComponent

```

public class **ProgressComponent** extends Component

ProgressComponent is a component used to represent the progress status. There are two types of ProgressComponent: Interactive and Non-interactive. They are determined at the time of creation. For the interactive type, the user key input is received and increased or decreased by the value defined by setStep. On the other hand, for the non-interactive type, the user key input is not received, and the value is changed by setValue. By default, step value is 1, although this can be changed using the setStep method. The minimum value for ProgressComponent is fixed as 0; only the maximum value is subject to change.

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

### Detailed Description of the Generator

- **ProgressComponent**

***public ProgressComponent(boolean blInteractive, int max)***

Generates new ProgressComponent with generator method 0 as the minimum value

**Parameters**

blInteractive	Sets whether or not Progress receives user input
max	Sets the maximum value for Progress

### Detailed Description of the Method

- **setStep**

***public void setStep(int step)***

This method sets the unit of increase or decrease for Progress Bar. It is used for the interactive type, since the increase or decrease is determined by user input. Input by setValue is also increased or decreased according to the given unit. When the step value is changed, the current value is also changed to step unit. The default value is 1.

**Parameters**

nStep	Amount of increase
-------	--------------------

**Throws**

IllegalArgumentException	Issued when the step value is 0 or larger than the maximum value
--------------------------	--

**Reference Item**

getStep()

- **getStep**

***public int getStep()***

Gets the unit of increase or decrease for Progress Bar

**Return Value**

Amount of increase set

**Reference Item**

#setStep()

- **setMargin**

***public void setMargin(int top, int bottom)***

Sets the top and bottom margins for Progress Bar (in pixel)

**Reference Item**

#setStep()

- **setMaxValue**

***public void setMaxValue(int maxValue)***

Sets the maximum value for Progress (in case of a current value that is larger than the maximum value given, the current value becomes the maximum value set)

**Parameters**

maxValue      Maximum value to be set

**Throws**

IllegalArgumentException      Issued when maxValue is 0 or less

- **setValue**

***public int setValue(int value)***

This method sets the current value for Progress. When the step is set by setStep, the value is changed by the step unit set. The result value is applied in Result value = value - value % step formula. Any value that is smaller than 0 is set as 0. In case the value is larger than MAX, however, it is set as MAX.

**Parameters**

value      Value to be set

**Return Value**

Value set

- **getValue**

***public int getValue()***

Used to get the current value set for Progress

**Return Value**

Current value set

- **getMaxValue**

***public int getMaxValue()***

Used to get the maximum value set

**Return Value**

Maximum value set

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **getPreferredWidth**

***public int getPreferredWidth()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The value returned by this method is referred to when Container determines the size of the component.

**Overrides**

getPreferredWidth in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Width of the component

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g Graphics for painting

**Reference Item**

Graphics

- **keyNotify**

***public boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the Component class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

keyNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **setChangeListener**

***public void setChangeListener(ChangeListener listener, Object obj)***

This method registers ChangeListener in ProgressComponent. In case of a value change in ProgressComponent, ChangeListener.changed(Component cmp, Object o) of ChangeListener is executed.

**Parameters**

l	ChangeListener to be registered in ListComponent
o	Object to be registered in ListComponent together with ChangeListener

### Class ScrollbarComponent

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ScrollbarComponent

```

public class **ScrollbarComponent** extends Component

ScrollBarComponent is a component with maximum and minimum values. It can change the value flexibly within that range.

In addition, ScrollBarComponent provides two types of scroll direction value: Horizontal and Vertical. This value can be specified using the `setDirection(int direction)` method. When a value other than Horizontal and Vertical is specified, `IllegalArgumentException` is issued.

The value for a scroll bar can be changed by the user key input or by methods such as `setCurrentValue`.

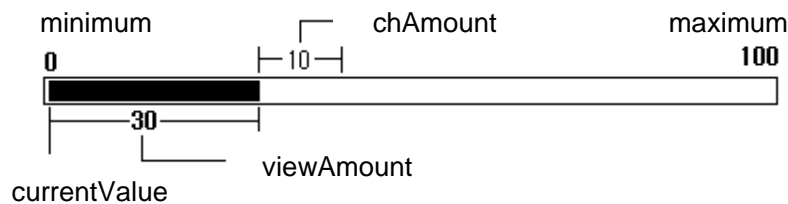


Figure 3-1-4-1. Example of a scroll bar value setting

For example, when creating ScrollbarComponent with size values as shown in Figure 3-1-4-1, use the `ScrollbarComponent(int direction, int currentValue, int viewAmount, int minimum, int maximum, int chAmount)` generator and they are specified as follows:

```
scrollBar = new ScrollbarComponent(HORIZONTAL, 0, 30, 0, 100,10);
```

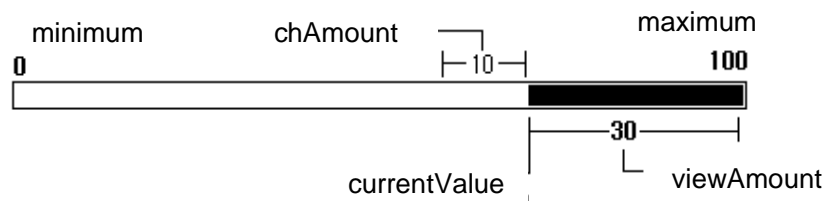


Figure 3-1-4-2. Example of a maximum current value setting for a scroll bar



As shown in Figure 3-1-4-2, the maximum value that can be specified for the current value is not the maximum value for the scroll bar; instead, it is a value obtained by deducting the view amount from the maximum value for the scroll bar. The size for each attribute of a scroll bar is as follows:

Table 3-1-4-1. Attribute size of a scroll

	Description	Default	Area
Direction	Scroll direction Vertical and horizontal values can be specified. When a value other than these two is specified, <code>IllegalArgumentException</code> is issued.	Vertical	Vertical Horizontal
currentValue	Current position value for a <code>ScrollbarComponent</code> Note that the view by which the current value can be actually specified is not the maximum value; instead, it is a value obtained by deducting <code>viewAmount</code> from the maximum value.	0	Minimum <= <code>currentValue</code> <= (Maximum - <code>viewAmount</code> )
viewAmount	Size of the area currently in use out of the total area	1	$0 < \text{viewAmount} \leq (\text{Maximum} - \text{Minimum})$
Minimum	Minimum value of <code>ScrollbarComponent</code>	0	Minimum < Maximum
Maximum	Maximum value of <code>ScrollbarComponent</code>	10	Minimum < Maximum
chAmount	Size of a scroll bar movement effected when a move action such as an arrow key input occurs in <code>ScrollbarComponent</code> .	1	$0 < \text{chAmount} \leq \text{viewAmount}$

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **HORIZONTAL**

***public static final int HORIZONTAL***

Direction value for horizontal scroll (set to 1)

- **VERTICAL**

***public static final int VERTICAL***

Direction value for vertical scroll (set to 2)

## Detailed Description of the Generator

- **ScrollbarComponent**

### ***public ScrollbarComponent()***

This method generates an instance of the scroll bar.

VERTICAL is the direction value for the vertical scroll, and it is specified as a default value. For the size of each attribute in ScrollbarComponent, the minimum value of 0 and maximum value of 10 are specified. The value for the starting position is 0. Both the view amount and change amount are set to 1.

#### **Reference Item**

ScrollbarComponent(int direction), ScrollbarComponent(int direction, int currentValue, int viewAmount, int minimum, int maximum, int chAmount)

- **ScrollbarComponent**

### ***public ScrollbarComponent(int direction)***

This method generates an instance of the scroll bar using the direction value of a specified scroll bar.

Horizontal and Vertical are the direction values of the scroll bar that can be specified. When a value other than these two is specified, IllegalArgumentException is issued.

For the size of each attribute in ScrollbarComponent, a minimum value of 0 and a maximum value of 10 are specified. The value for the starting position is 0. Both the view amount and change amount are set to 1.

#### **Parameters**

direction          Direction value of the scroll bar

#### **Throws**

IllegalArgumentException	Issued when a value other than Horizontal and Vertical is specified as the direction value of the scroll bar
--------------------------	--

#### **Reference Item**

ScrollbarComponent(), ScrollbarComponent(int direction, int currentValue, int viewAmount, int minimum, int maximum, int chAmount)

- **ScrollbarComponent**

***public ScrollbarComponent(int direction, int currentValue, int viewAmount, int minimum, int maximum, int chAmount)***

This method generates an instance of ScrollbarComponent using the size of each attribute and direction values of a given scroll.

Horizontal and Vertical are the direction values of the scroll that can be specified. When a value other than these two is specified, IllegalArgumentException is issued.

The size of each attribute in ScrollbarComponent should be specified such that it fits the area described in Attribute Size of the scroll. When the value specified is out of bounds, IllegalArgumentException is issued.

**Parameters**

direction	Direction value of the scroll bar
currentValue	Position value of the current scroll bar
viewAmount	View amount of the scroll bar
minimum	Minimum value of the scroll bar
maximum	Maximum value of the scroll bar

**Throws**

IllegalArgumentException	Issued when a value other than Horizontal and Vertical is specified as the direction value of the scroll bar
IllegalArgumentException	Issue when an error occurs in each attribute value

**Reference Item**

ScrollbarComponent(), ScrollbarComponent(int direction)

### Detailed Description of the Method

- **getDirection**

***public int getDirection()***

Returns the scroll direction value of a currently specified ScrollbarComponent; when a specific scroll direction value is not specified, the default scroll direction value is Vertical

**Return Value**

Direction value for the scroll

**Reference Item**

setDirection(int direction)

- **setDirection**

***public void setDirection(int direction)***

This method sets the scroll direction value for a scroll bar.

Horizontal and Vertical are the direction values that can be specified. When a value other than these two is specified, IllegalArgumentException is issued.

**Parameters**

direction          Direction value of the scroll

**Throws**

IllegalArgumentException      Issued when a value other than Horizontal or  
Vertical is specified for the direction value of the  
scroll bar

**Reference Item**

getDirection()

- **getCurrentValue**

***public int getCurrentValue()***

Returns currentValue of a scroll bar

**Return Value**

currentValue

**Reference Item**

setCurrentValue(int newValue)

- **setCurrentValue**

***public void setCurrentValue(int newValue)***

This method specifies currentValue such that it fits the area described in Attribute Size of the scroll. When the value specified is smaller than the minimum value or larger than the maximum value, the value is set to the maximum value or the minimum value.

Note that the maximum value that can be used to specify the current value is not the maximum value of the scroll bar; instead, it is a value obtained after deducting the view amount from the maximum value as shown in the figure.

**Parameters**

newValue      Current scroll value to be specified

**Reference Item**

getCurrentValue()

- **getMinimum**

***public int getMinimum()***

Returns the minimum value of a scroll bar

**Return Value**

Minimum value of the scroll bar

**Reference Item**

setMinimum(int newMinimum)

- **setMinimum**

***public void setMinimum(int newMinimum)***

This method specifies the minimum value such that it fits the area described in Attribute Size of the scroll. The current value can be changed as the minimum value changes.

**Parameters**

newMinimum Minimum value that can be scrolled

**Reference Item**

getMinimum()

- **getMaximum**

***public int getMaximum()***

Returns the maximum value of a scroll bar

**Return Value**

Maximum value of the scroll bar

**Reference Item**

setMaximum(int newMaximum)

- **setMaximum**

***public void setMaximum(int newMaximum)***

This method specifies the maximum value such that it fits the area described in Attribute Size of the scroll. The current value can be changed as the maximum value changes.

Note that the maximum value that can be used to specify the current value is not the maximum value of the scroll bar; instead, it is a value obtained after deducting the view amount from the maximum value as shown in the figure.

**Parameters**

newMaximum Maximum value of the scroll bar

**Reference Item**

getMaximum()

- **getViewAmount**

***public int getViewAmount()***

Gets the view amount of a scroll bar

**Return Value**

viewAmount

**Reference Item**

setViewAmount(int newAmount)

- **setViewAmount**

***public void setViewAmount(int newAmount)***

This method specifies the view amount such that it fits the area described in Attribute Size of the scroll.

Note that the maximum value that can be used to specify the current value is not the maximum value of the scroll bar; instead, it is a value obtained after deducting the view amount from the maximum value as shown in the figure.

**Parameters**

newAmount Amount of data shown on the display

**Reference Item**

getViewAmount()

- **getChangeAmount**

***public int getChangeAmount()***

Gets the change amount of a scroll

**Return Value**

changeAmount

**Reference Item**

setChangeAmount(int newChAmount)

- **setChangeAmount**

***public void setChangeAmount(int newChAmount)***

Sets the change amount such that it fits the area described in Attribute Size of the scroll

**Parameters**

NewChAmount      Change amount of the scroll

**Reference Item**

getChangeAmount()

- **getForegroundColor**

***public int getForegroundColor()***

This method returns the foreground color of a scroll bar. If it is not specified, the default value is 0 x 00000000.

**Return Value**

Foreground color of the current scroll bar

**Reference Item**

setForegroundColor(int)

- **setForegroundColor**

***public void setForegroundColor(int fg)***

This method specifies the foreground color of a scroll bar, which is 0 x 00RRGGBB. The default foreground color is 0 x 00000000.

**Parameters**

fg      Value for the foreground color

**Reference Item**

getForegroundColor()



- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

g Graphics for painting

**Reference Item**

Graphics

- **focusNotify**

***public void focusNotify(boolean b)***

The following explanation of this method is copied from the Component class

This method is called when it receives a focus. To show whether or not the component has a focus, this method calls the repaint method to redraw itself.

**Overrides**

focusNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

b True when the component has a focus; otherwise, False is returned

- **keyNotify**

***public boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the Component class

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

keyNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **getPreferredHeight**

***public int getPreferredHeight()***

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Copied from class: org.kwis.msp.lwc.Component

**Parameters**

w     Variable width

**Return Value**

Height of the component

- **getPreferredWidth**

***public int getPreferredWidth()***

The following explanation of this method is copied from the Component class.

This method determines the preferred width of the component. The value returned by this method is referred to when Container determines the size of the component.

**Overrides**

getPreferredWidth in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Width of the component

## Class ShellComponent

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.ContainerComponent
|
+--org.kwis.msp.lwc.ShellComponent

```

Directly known subclasses:  
AnnunciatorComponent, DialogComponent

public class **ShellComponent** extends ContainerComponent

ShellComponent provides a connection to Card. It has a title, a command input component, and a work component. This component should be used at the very top to show UI components on the display. ShellComponent can have only one work component through AddComponent. This component makes a connection to lcdui Card and transmits the incoming event from Card to the relevant component.

This component also shows the title and frames on the display.

### Fields inherited from class org.kwis.msp.lwc.ContainerComponent

cmpFocus, cmps, insetBottom, insetLeft, insetRight, insetTop, ncomp, offsetX, offsetY, useFrame

### Fields inherited from class org.kwis.msp.lwc.Component

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

### Methods inherited from class org.kwis.msp.lwc.ContainerComponent

getComponent, getIndexOf, getNumberOfComponent, paint, paintFrame, removeComponent, repaint, scrollTo, setComponent, useFrame, validate

**Methods inherited from class org.kwis.msp.lwc.Component**

CalcPreferredSize, canHandleInput, focusNotify, getBackground, getForeground, getHeight, getPreferredSize, getPreferredSize, getPreferredSize, getWidth, getXOnScreen, getYOnScreen, hasFocus, invalidate, isValid, paintContent, pointerNotify, setBackground, setEventListener, setFocus, setForeground, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Field**

- **cd**

***protected Card cd***

Field that stores a connected card

- **cmpTitle**

***protected Component cmpTitle***

Component that becomes the title

- **cmpWork**

***protected Component cmpWork***

Component that is located between the title and a command

- **cmpCommand**

***protected Component cmpCommand***

Command component

**Detailed Description of the Generator**

- **ShellComponent**

***public ShellComponent()***

Generates ShellComponent of the display size

**Parameters**

- **ShellComponent**

***public ShellComponent(boolean inflate)***

Generates ShellComponent and determines whether the size of ShellComponent should fit the full display or an internal component depending on inflate

**Parameters**

inflate Whether ShellComponent should be enlarged to fit a component

- **ShellComponent**

***public ShellComponent(int x, int y, int w, int h)***

Generates ShellComponent with a specified size

**Parameters**

x Position on the component display  
 y Position on the component display  
 w Width on the component display  
 h Height on the component display

- **ShellComponent**

***public ShellComponent(int x, int y, int w, int h, boolean bTrans)***

This method generates a ShellComponent with a specified size. Depending on bTrans, this ShellComponent can become a transparent shell.

**Parameters**

x Position on the component display  
 y Position on the component display  
 w Width on the component display  
 h Height on the component display  
 bTrans Whether or not the component is transparent

### Detailed Description of the Method

- **layout**

***public void layout()***

The following explanation of this method is copied from the Component class.

Determines the size and position of a child component

**Overrides**

layout in class Component

- **repaint**

***public void repaint(int x, int y, int w, int h)***

The following explanation of this method is copied from the **Component** class.

This method is called when there is a need to override the display content. It finally calls the repaint of the card, and the called repaint method automatically undergoes a process of calling the paint() method of the component after a certain period of time.

**Overrides**

repaint in class ContainerComponent

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

x	X coordinate of the area to be overridden
y	Y coordinate of the area to be overridden
w	Width of the area to be overridden
h	Height of the area to be overridden

- **show**

***public void show()***

This method shows a component on the display. Before showing it on the display, this method calculates the position and size of the component using the validate method.

- **hide**

***public void hide()***

Hides a component

- **isShown**

***public boolean isShown()***

The following explanation of this method is copied from the **Component** class.

This method indicates whether or not the current component is shown. True is returned when it is shown on the display; otherwise, False is returned.

**Overrides**

isShown in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Whether or not the current component is shown on the display

- **configure**

***public void configure(int x, int y, int w, int h, int mask)***

The following explanation of this method is copied from the Component class.

This method changes the position and size of a component depending on the mask. A logical AND operation is carried out for the mask value and POS\_MASK. This method changes a value of POS\_MASK to positions x and y in the parent component. A logical AND operation is also carried out for the mask value and SIZE\_MASK. If the value is SIZE\_MASK, this method changes the size of the component to (w, h). In other words, the size and position of the component can be changed at the same time. This method calls the repaint method for the changed area, causing the area to be painted by the paintContent method. The size of the component is determined by the layout method of the parent component.

**Overrides**

configure in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

x	X coordinate of the component on the parent component
y	Y coordinate of the component on the parent component
w	Width of the component
h	Height of the component
mask	POS_MASK   SIZE_MASK; when POS_MASK is received, x and y values become valid values (when SIZE_MASK is received, w and h values become valid values)

- **getX**

***public int getX()***

The following explanation of this method is copied from the Component class.

Returns the x coordinate of the component on the parent component

**Overrides**

getX in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

X coordinate



- **getY**

***public int getY()***

The following explanation of this method is copied from the Component class.

Returns the y coordinate of the component on the parent component

**Overrides**

getY in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Y coordinate

- **showNotify**

***public void showNotify(boolean b)***

The following explanation of this method is copied from the Component class.

- **showNotify**

***protected void showNotify(boolean bShow)***

This method is called when the content of the display is shown. It is called by addComponent or removeComponent or when the top-most parent component (ShellComponent) is shown on the display by show.

**Overrides**

showNotify in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

bShow Whether or not the component is shown

- **serviceRepaints**

***public void serviceRepaints()***

The following explanation of this method is copied from the Component class.

This method prints the repainted content immediately on the display. Instead of calling paint later through repaint, this method calls the paint method directly to print it on the display.

**Overrides**

serviceRepaints in class Component

- **addComponent**

***public void addComponent(int index, Component cmp)***

The following explanation of this method is copied from the **ContainerComponent** class.

This method adds one child component indicated by *cmp* on the specified position.  
When *cmp* is null, **NullPointerException** is issued. On the other hand, when *cmp* has another parent component, **IllegalArgumentException** is issued.  
In case of an index value that is smaller than 0 or larger than the number of added components, **IndexOutOfBoundsException** is issued.

**Overrides**

addComponent in class **ContainerComponent**

Following copied from class: `org.kwis.msp.lwc.ContainerComponent`

**Parameters**

index    Position where a component is to be added

cmp      Component to be added

**Throws**

**IllegalArgumentException**      Issued when *cmp* has another parent component  
cmp

**IndexOutOfBoundsException**    Issued when an invalid index is specified

**NullPointerException**          Issued when *cmp* is null

- **addComponent**

***public int addComponent(Component cmp)***

The following explanation of this method is copied from the **ContainerComponent** class.

Adds a child component at the very top

**Overrides**

addComponent in class **ContainerComponent**

Following copied from class: `org.kwis.msp.lwc.ContainerComponent`

**Parameters**

cmp      Child component to be added

**Throws**

**IllegalArgumentException**      Issued when *cmp* has another parent component

**NullPointerException**          Issued when *cmp* is null

- **processEvent**

***protected boolean processEvent(int type, int subtype, int param1, int param2)***

The following explanation of this method is copied from the Component class.

Processes an event

**Overrides**

processEvent in class ContainerComponent

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

type	Event type
subtype	Sub-event type subject to an event type
param1	Additional argument
param2	Additional argument

- **setTitle**

***public void setTitle(Component cmp)***

Prints a title on a specific label component

**Parameters**

cmp	Label component to be specified as the title
-----	--

- **getTitle**

***public Component getTitle()***

Returns a specified title

**Return Value**

Title component

- **getWorkComponent**

***public Component getWorkComponent()***

Returns to a specified component

**Return Value**

Work component added

- **setWorkComponent**

***public void setWorkComponent(Component cmp)***

Sets a work component

**Parameters**

cmp    Work component to be set

- **removeAllComponents**

***public void removeAllComponents()***

The following explanation of this method is copied from the ContainerComponent class.

Removes all components

**Overrides**

removeAllComponents in class ContainerComponent

- **removeComponent**

***public void removeComponent(Component cmp)***

The following explanation of this method is copied from the ContainerComponent class.

This method removes a specified component. When the specified component is not registered as a child component, IllegalArgumentException is issued.

**Overrides**

removeComponent in class ContainerComponent

Following copied from class: org.kwis.msp.lwc.ContainerComponent

**Parameters**

cmp    Component to be removed

- **setCommand**

***public void setCommand(Component cmp, boolean bGrab)***

This method sets the command that appears at the bottom of a display. The component can have the keyNotify() method called for all key events. This method operates when bGrab is True.

**Parameters**

cmp    Component that is to become a command

bGrab   All keys grabbed by this component preferentially

- **getCommand**

***public Component getCommand()***

Returns a specified command component

**Return Value**

Specified command component

- **getNextTraversalComponent**

***protected Component getNextTraversalComponent()***

The following explanation of this method is copied from the **ContainerComponent** class.

This method returns the next traversal component. The component to be returned will be one of the child components of the current component. In the absence of a traversal component, a null value is returned.

**Overrides**

getNextTraversalComponent in class ContainerComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Return Value**

Next traversal component

- **getPrevTraversalComponent**

***protected Component getPrevTraversalComponent()***

The following explanation of this method is copied from the **ContainerComponent** class.

This method returns the previous traversal component. The component to be returned will be one of the child components of the current component. In the absence of a traversal component, a null value is returned.

**Overrides**

getPrevTraversalComponent in class ContainerComponent

Copied from class: org.kwis.msp.lwc.ContainerComponent

**Return Value**

Previous traversal component

- **setTitle**

***public void setTitle(String str)***

This method sets a title string. It sets specific characters to be printed as a title.

**Parameters**

str      String to be set as a title

- **getCard**

***public Card getCard()***

The following explanation of this method is copied from the Component class.

This method returns the card that is connected to the current component. A parent component like ShellComponent has an internal card. This method returns the card of the top-most parent component of the component. In the absence of the parent component, however, a null value can be returned.

**Overrides**

getCard in class Component

Copied from class: org.kwis.msp.lwc.Component

**Return Value**

Card that is connected to the component

**Reference Item**

Card

- **grabKey**

***public void grabKey(int key)***

This method grabs a specific key code and sends it to GrabKeyListener. When grabbing a specific key using this method, an event related to the specific key is first sent to GrabKeyListener for processing instead of sending it to a child component directly.

To get an undefined key code, GameKey is returned using the Display.getKeyCode() method.

**Parameters**

key     Key code to be grabbed (see Event Queue)

**Reference Item**

setGrabKeyListener(org.kwis.msp.lwc.GrabKeyListener, java.lang.Object),

ungrabKey(int), GrabKeyListener, Display.getKeyCode(int)

- **ungrabKey**

***public void ungrabKey(int key)***

Releases grab for a specific key code

**Parameters**

gameKey      Key code to be ungrabbed (see the key code of EventQueue)

**Reference Item**

grabKey(int), setGrabKeyListener(org.kwis.msp.lwc.GrabKeyListener,  
java.lang.Object), GrabKeyListenerDisplay.getKeyCode(int)

- **setGrabKeyListener**

***public void setGrabKeyListener(GrabKeyListener listener, Object obj)***

This method registers GrabKeyListener. When a grab is set by grabKey, this method registers a listener to receive a grabbed key event. The key event is first sent to a specified event listener. When True is returned, the event is not processed. On the other hand, the event is processed when the event listener returns False.

**Parameters**

listener      ey grab listener  
obj      arameter when the key grab listener is called

**Reference Item**

GrabKeyListener, grabKey(int), ungrabKey(int)

- **keyNotify**

***protected boolean keyNotify(int type, int chr)***

**The following explanation of this method is copied from the Component class.**

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is chr.

**Overrides**

keyNotify in class ContainerComponent

Following copied from class: org.kwis.msp.lwc.Component



**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
chr	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

**● controllInset*****protected void controllInset(boolean flag)***

The following explanation of this method is copied from the **ContainerComponent** class.

This method controls the thickness value of a frame used in a **ContainerComponent**. When True is returned, the thickness value of the frame specified in each **ContainerComponent** is applied. Otherwise, when False is returned, the thickness value of the currently specified frame is initialized to 0.

**Overrides**

controllInset in class **ContainerComponent**

### Class TextBoxComponent

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.TextComponent
|
+--org.kwis.msp.lwc.TextBoxComponent

```

```
public class TextBoxComponent extends TextComponent
```

This is a class that inherited TextComponent. TextBoxComponent can edit characters to fit the specified width. The width of this component is the same as the width of ContainerComponent that added this component. The height is automatically changed to fit the character data printed on the current display.

TextBoxComponent can edit characters using the full display and can limit inputs to specified strings (refer to the constraint described in TextComponent).

Basically, there is no limit to the size of an input string, although the number of allowable input characters can be limited using TextComponent.setMaxLength(int maxLen).

#### ● Reference Item

TextComponent, TextFieldComponent

#### Fields inherited from class org.kwis.msp.lwc.TextComponent

charCount, constChecker, constraint, CONSTRAINT\_ANY, CONSTRAINT\_NUMBER, CONSTRAINT\_PASSWORD, display, f, imHandler, iMode, isWide, m\_cPos, m\_td, maxLength, modeViewer, tShell

#### Fields inherited from class org.kwis.msp.lwc.Component

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.TextComponent**

FocusNotify, getConstraint, getFont, getMaxLength, getString, setMaxLength, showNotify

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **TextBoxComponent**

***public TextBoxComponent(String data, int constraint)***

This method generates an instance of TextComponent using a given character data and constraints. The character data can have a null value.

When a value other than TextComponent.CONSTRAINT\_NUMBER, TextComponent.CONSTRAINT\_PASSWORD, or TextComponent.CONSTRAINT\_ANY is specified as a constraint, IllegalArgumentException is issued. On the other hand, when character data includes data that do not comply with the constraint, IllegalArgumentException is issued.

Basically, there is no limit to the size of an input string, although the number of allowable input characters can be limited using TextComponent.setMaxLength(int maxLen).

**Parameters**

data	haracter data; may also be a null value
constraint	Character input type

- **TextBoxComponent**

***public TextBoxComponent(String data, int constraint, int h)***

This method generates an instance of TextComponent using a given character data, constraints, and height value of the component. The character data can be null. The specified height value is the height value of the initial component. When the total height value becomes larger than the specified height value due to an increase in data within the component, the height of the current component will be replaced by a larger value.

When a value other than TextComponent.CONSTRAINT\_NUMBER, TextComponent.CONSTRAINT\_PASSWORD, or TextComponent.CONSTRAINT\_ANY is specified as a constraint, IllegalArgumentException is issued. On the other hand, when character data includes data that does not comply with the constraint, IllegalArgumentException is issued.

Basically, there is no limit to the size of an input string, although the number of allowable input characters can be limited using TextComponent.setMaxLength(int maxLen).

**Parameters**

data	Character data; may also be a null value
constraint	Character input type
h	Height value of the component

**Detailed Description of the Method**

- **setString**

***public void setString(String data)***

The following explanation of this method is copied from the TextComponent class.

This method sets character data. When a constraint is specified, IllegalArgumentException is issued if the character data is not a string that complies with the constraint.

When there is a limit to the number of allowable input characters, and if the number of characters inputted exceeds the maximum number of allowable input characters, only the maximum number of allowable input characters is processed as its own data.

When the character data is null, the current character data is changed to "" or an empty string. Therefore, no string will be shown on the display.

**Overrides**

setString in class TextComponent

Following copied from class: org.kwis.msp.lwc.TextComponent

**Parameters**

data	Character data; may also be a null value
------	--

**Throws**

IllegalArgumentException	Issued when a string that complies with the current
--------------------------	---

constraint is not specified

### Reference Item

TextComponent.getString()

### ● insert

***public void insert(char[] data, int offset, int len, int index)***

The following explanation of this method is copied from the TextComponent class.

This method adds the character data given as an argument in the character data printed on the current display to the index position.

When the data is null, NullPointerException is issued. In case of an index value that is smaller than 0 or larger than the total length of the character data printed on the current display, IndexOutOfBoundsException is issued.

On the other hand, when there is a limit to the number of allowable input characters, and if the aggregate of the length of the character data currently printed and len is larger than the maximum number of allowable input characters, IllegalArgumentException is issued.

### Overrides

Insert in class TextComponent

Following copied from class: org.kwis.msp.lwc.TextComponent

### Parameters

data	Character data to be added
offset	Starting position of the character data to be added
len	Length of the character data to be added
index	Position where new character data is to be added in the character data printed on the current display

### Throws

NullPointerException	Issued when the data is null
IndexOutOfBoundsException	Issued in case of an index value that is smaller than 0 or larger than the total length of the character data printed on the current display
IllegalArgumentException	Issued when there is a limit to the number of allowable input characters, and if the aggregate of the length of the character data currently printed and len is larger than the maximum number of allowable input

	characters
IllegalArgumentException	Issued when the character data to be inputted includes character data that does not comply with the current constraint

- **delete**

***public void delete(int index, int len)***

The following explanation of this method is copied from the **TextComponent** class.

Data as long as len from the index position of the current character data shown on the display is deleted.

In case of an index value that is smaller than 0 or larger than the total length of the character data printed on the current display, **IndexOutOfBoundsException** is issued.

When len exceeds the total length of the character data currently printed, **IllegalArgumentException** is issued.

**Overrides**

Delete in class **TextComponent**

Following copied from class: `org.kwis.msp.lwc.TextComponent`

**Parameters**

index    Position where the data is to be deleted

len      Length of the character data to be deleted

**Throws**

<b>IndexOutOfBoundsException</b>	Issued in case of len or index value that is smaller than 0 or larger than the total length of the character data printed on the current display
----------------------------------	--

<b>IllegalArgumentException</b>	Issued when len exceeds the total length of the character data currently printed
---------------------------------	--

- **getPreferredWidth**

***public int getPreferredWidth()***

The following explanation of this method is copied from the **Component** class.

This method determines the preferred width of the component. The value returned by this method is referred to when **Container** determines the size of the component.

**Overrides**

getPreferredWidth in class **Component**

Following copied from class: `org.kwis.msp.lwc.Component`

**Return Value**

Width of the component

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **configure**

***public void configure(int x, int y, int w, int h, int mask)***

The following explanation of this method is copied from the Component class.

This method changes the position or size of a component depending on the mask. A logical AND operation is carried out for the mask value and POS\_MASK. This method changes a value of POS\_MASK to positions x and y in the parent component. A logical AND operation is also carried out for the mask value and SIZE\_MASK. If the value is SIZE\_MASK, this method changes the size of the component to (w, h). In other words, the size and position of the component can be changed at the same time. This method calls the repaint method for the changed area, causing the area to be painted by the paintContent method. The size of the component is determined by the layout method of the parent component.

#### Overrides

Configure in class Component

Following copied from class: org.kwis.msp.lwc.Component

#### Parameters

x	X coordinate of the component on the parent component
y	Y coordinate of the component on the parent component
w	Width of the component
h	Height of the component
mask	POS_MASK   SIZE_MASK; when POS_MASK is received, x and y values become valid values (when SIZE_MASK is received, w and h values become valid values)

#### ● keyNotify

***public boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the TextComponent class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is ch.

(EventQueue) When the SOFT1 key is pressed, the input mode can be changed to input mode. The input mode is in charge of executing the notifykeyInput method of InputMethodHandler, which takes care of character handling for the input of number keys. It also has control over the character editing using the full display.

#### Overrides

keyNotify in class TextComponent

Following copied from class: org.kwis.msp.lwc.TextComponent

#### Parameters

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a
------	--



key is pressed and held; and KEY\_TYPED when a key is pressed once and released

key      Character of the key pressed; default values are numbers 0 to 9 and symbols \* and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

- **setFont**

***public void setFont(Font f)***

The following explanation of this method is copied from the TextComponent class.

Sets the font (the font is set using Font.getDefaultFont() as a default value)

**Overrides**

setFont in class TextComponent

Following copied from class: org.kwis.msp.lwc.TextComponent

**Parameters**

f      Font to be set

**Reference Item**

TextComponent.getFont()

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

g      Graphics for painting

**Reference Item**

Graphics

**Class TextComponent**

```
java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.TextComponent
```

Directly known subclasses:  
TextBoxComponent, TextFieldComponent

public abstract class **TextComponent** extends Component

This is an abstract class that is used to print as well as to input, modify, and delete text.

The classes that are implemented by inheriting this class include TextBoxComponent and TextFieldComponent.

#### Character Editing Using the Full Display

In case of the EventQueue.FIRE input in TextFieldComponent or TextBoxComponent, an editor that can be used for character editing using the full display is created. When editing is completed in this editor, the previous display is shown by entering the EventQueue.FIRE key. The character data of the previous display is changed to the character data that is edited on the full display.

#### Inputting Constraint

TextComponent provides six types of constraints:

CONSTRAINT\_NUMBER is a constraint that allows only “-,”,” and number inputs.

CONSTRAINT\_PASSWORD is a constraint for inputting passwords. The string used internally allows only numbers. The type printed on the display is “\*.”

CONSTRAINT\_EMAILADDRESS is a constraint for inputting e-mail addresses. The string used internally includes English characters (both upper and lower cases), numbers, and symbols.

CONSTRAINT\_URL is a constraint for inputting URLs. The string used internally includes English characters (both upper and lower cases), numbers, and symbols.

CONSTRAINT\_PHONENUMBER is a constraint for inputting phone numbers. The string used internally is numbers.

CONSTRAINT\_ANY is \*, which is a constraint for inputting any kind of string.

Only the constraints described above can be specified. When a value other than what is described above is specified as a constraint, IllegalArgumentException is issued.

Basically, there is no limit to the size of an input string, although the number of allowable input characters can be limited using `setMaxLength(int maxLen)`.

The implementation of `InputMethodListener` and `ActionListener` used for processing character inputs includes internal classes. Each internal class manages character inputs based on the current input mode, converts the input display to full display, or restores them to their original display, i.e., `TextFieldComponent` or `TextBoxComponent`.

## ● Reference Item

`TextFieldComponent`, `TextBoxComponent`

### Fields inherited from class `org.kwis.msp.lwc.Component`

`bg`, `evtListener`, `evtListenerObj`, `fg`, `FOCUS_NOTIFY`, `h`, `HAS_FOCUS_MASK`, `INPUT_MASK`, `KEY_NOTIFY`, `KEY_PRESSED`, `KEY_RELEASED`, `KEY_REPEATED`, `KEY_TYPED`, `LAYOUT_BOTTOM`, `LAYOUT_HCENTER`, `LAYOUT_LEFT`, `LAYOUT_RIGHT`, `LAYOUT_TOP`, `LAYOUT_VCENTER`, `mask`, `parent`, `POINT_DRAGGED`, `POINT_PRESSED`, `POINT_RELEASED`, `POINTER_NOTIFY`, `POS_MASK`, `PREFER_SIZE_MASK`, `prefH`, `prefW`, `SHOW_NOTIFY`, `SIZE_MASK`, `VALID_MASK`, `w`, `x`, `y`

### Methods inherited from class `org.kwis.msp.lwc.Component`

`calcPreferredSize`, `canHandleInput`, `configure`, `getBackground`, `getCard`, `getForeground`, `getHeight`, `getPreferredSize`, `getPreferredSize`, `getPreferredSize`, `getWidth`, `getX`, `getXOnScreen`, `getY`, `getYOnScreen`, `hasFocus`, `invalidate`, `isShown`, `isValid`, `layout`, `paintContent`, `pointerNotify`, `processEvent`, `repaint`, `repaint`, `serviceRepaints`, `setBackground`, `setEventListener`, `setFocus`, `setForeground`, `toString`, `validate`

### Methods inherited from class `java.lang.Object`

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

### Detailed Description of the Field

#### ● CONSTRAINT\_NUMBER

***public static final int CONSTRAINT\_NUMBER***

This is a constraint that is used when allowable input characters are limited to " , " - , " and numbers. The value assigned to `CONSTRAINT_NUMBER` is 1.

- **CONSTRAINT\_PASSWORD**

- public static final int CONSTRAINT\_PASSWORD***

- This is a constraint that is used when allowable input characters are limited to passwords. The string used internally is numbers, and the type printed on the display is ".\*." The value assigned to CONSTRAINT\_PASSWORD is 2.

- **CONSTRAINT\_ANY**

- public static final int CONSTRAINT\_ANY***

- This is a constraint that is used when there is no limit to the allowable input characters. This is used as a default value in the absence of a specified constraint. The value assigned to CONSTRAINT\_ANY is 0.

- **CONSTRAINT\_EMAILADDRESS**

- public static final int CONSTRAINT\_EMAILADDRESS***

- This is a constraint that is used when the allowable input characters are limited to e-mail addresses. The string used internally includes English characters (both upper and lower cases), numbers, and symbols. The value assigned to CONSTRAINT\_EMAILADDRESS is 3.

- **CONSTRAINT\_URL**

- public static final int CONSTRAINT\_URL***

- This is a constraint that is used when allowable input characters are limited to URLs. The string used internally includes English characters (both upper and lower cases), numbers, and symbols. The value assigned to CONSTRAINT\_URL is 4.

- **CONSTRAINT\_PHONENUMBER**

- public static final int CONSTRAINT\_PHONENUMBER***

- This is a constraint that is used when allowable input characters are limited to phone numbers. The string used internally is numbers. The value assigned to CONSTRAINT\_PHONENUMBER is 5.

- **imHandler**

- protected InputMethodHandler imHandler***

- Input method handler that is used to process other character data for key inputs

- **m\_cPos**

- protected int m\_cPos***

- Position of the current character data

- **charCount**

- protected int charCount***

- Number of characters

- **constraint**

- protected int constraint***

- Input type specification

- **m\_td**

- protected char[] m\_td***

- Character data

- **iMode**

- protected int iMode***

- String input mode that is currently in use

- **display**

- protected Display display***

- Display object for card use

- **modeViewer**

- protected org.kwis.msp.lwc.TextComponent.ModeViewer modeViewer***

- Card showing the current input mode

- **isWide**

- protected boolean isWide***

- Character input status using the full display (full input display status when True is returned; default value is False)

- **tShell**

- protected ShellComponent tShell***

- Used when converting to the full display

- **maxLength**

- protected int maxLength***

- No limit to the length of allowable input characters when the value is -1

- **constChecker**

- protected org.kwis.msp.lwc.ConstraintChecker constChecker***

- Checks whether the character data fit the specified constraint

- **f**

- protected Font f***

- This is the font that is used in TextComponent. By default, this font is created through Font.getDefaultFont(). In case another font is preferred, the font can be changed through setFont(org.kwis.msp.lcdui.Font).

**Reference Item**

setFont(Font f)

**Detailed Description of the Method**● **setString*****public void setString(String data)***

This method sets the character data. When a constraint is specified, and if character data is not a string that complies with the constraint, `IllegalArgumentException` is issued. When there is a limit to the number of allowable input characters, and if the number of characters inputted exceeds the maximum number of allowable input characters, only the maximum number of allowable input characters is processed as its own data. When the character data is null, the current character data is changed to "" or an empty string. Therefore, no string will be shown on the display.

**Parameters**

data    Character data; may also be a null value

**Throws**

<code>IllegalArgumentException</code>	Issued when a string that complies with the current constraint is not specified
---------------------------------------	---

**Reference Item**

getString()

- **setMaxLength**

***public void setMaxLength(int maxLen)***

This method sets the maximum number of allowable input characters. The default value is -1, which does not put any limit to the number of allowable input characters.

When the length of the existing characters is longer than the length of the characters specified, only the characters that can be inputted in length is re-set as data. When there is a limit to the number of allowable input characters, input is possible only when the number of characters is smaller than the maximum number of allowable input characters. All other inputs are ignored.

When the maximum number of allowable input characters is 0 or less than -1, `IllegalArgumentException` is issued.

**Parameters**

maxLen	Maximum number of allowable input characters
--------	--

**Throws**

<code>IllegalArgumentException</code>	Issued when the maximum number of allowable input characters is 0 or less than -1
---------------------------------------	---

- **getMaxLength**

***public int getMaxLength()***

This method returns the maximum number of allowable input characters that are currently set. The default value is -1, which does not put any limit to the number of allowable input characters.

**Return Value**

Maximum number of allowable input characters

- **getString**

***public String getString()***

This method returns the current character data. When the current string is specified as a null value, "" (empty string) is returned.

**Return Value**

Character data

**Reference Item**

`setString(String data)`



- **insert**

***public void insert(char[] data, int offset, int len, int index)***

This method adds the character data given as an argument in the character data printed on the current display to the index position.

When the data is null, NullPointerException is issued. On the other hand, in case of an index value that is smaller than 0 or larger than the total length of the character data that is printed on the current display, IndexOutOfBoundsException is issued.

In addition, when there is a limit to the number of allowable input characters, and if the aggregate of the length of the character data currently printed and len is larger than the maximum number of allowable input characters, IllegalArgumentException is issued.

**Parameters**

data	Character data to be added
offset	Starting position of the character data to be added
len	Length of the character data to be added
index	Position where new character data is to be added in the character data printed on the current display

**Throws**

NullPointerException	Issued when the data is null
IndexOutOfBoundsException	Issued in case of an index value that is smaller than 0 or larger than the total length of the character data printed on the current display
IllegalArgumentException	Issued when there is a limit to the number of allowable input characters, and if the aggregate of the length of the character data currently printed and len is larger than the maximum number of allowable input characters
IllegalArgumentException	Issued when the character data to be inputted includes character data that does not comply with the current constraint

- **delete**

***public void delete(int index, int len)***

Data as long as len from the index position of the current character data being shown on the display is deleted.

In case of an index value that is smaller than 0 or larger than the total length of the character data that is printed on the current display, IndexOutOfBoundsException is issued. On the other hand, when len exceeds the total length of the character data

currently printed, `IllegalArgumentException` is issued.

#### Parameters

`index` Position where the data is to be deleted  
`len` Length of the character data to be deleted

#### Throws

<code>IndexOutOfBoundsException</code>	Issued in case of <code>len</code> or <code>index</code> value that is smaller than 0 or larger than the total length of the character data that is printed on the current display
<code>IllegalArgumentException</code>	Issued when <code>len</code> exceeds the total length of the character data currently printed

#### ● `getConstraint`

***`public int getConstraint()`***

Returns the currently specified character data constraint (the default restraint specified is `CONSTRAINT_ANY`)

#### Return Value

Constraint

#### ● `focusNotify`

***`public void focusNotify(boolean b)`***

This method is called when it receives a focus. When it receives a focus while it is shown on the display, this method prints the input mode card that indicates the input status on the current display. When it does not receive a focus, however, this method removes the input mode card from the display.

#### Overrides

`focusNotify` in class `Component`

#### Parameters

`b` True when it has focus; otherwise, `False` is returned

#### ● `showNotify`

***`protected void showNotify(boolean bShow)`***

The following explanation of this method is copied from the `Component` class.

This method is called when the content of the display is shown. This method is called by `addComponent` or `removeComponent`, or when the top-most parent component

(ShellComponent) is shown on the display by show.

### Overrides

showNotify in class Component

Copied from class: org.kwis.msp.lwc.Component

### Parameters

bShow Whether or not the component is shown

- **setFont**

***public void setFont(Font f)***

Sets the font (the font is set using Font.getDefaultFont() as a default value)

### Parameters

f Font to be set

### Reference Item

getFont()

- **getFont**

***public Font getFont()***

Gets the font

### Return Value

Font currently in use

### Reference Item

setFont(Font f)

- **keyNotify**

***public boolean keyNotify(int type, int key)***

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is ch.

(EventQueue) When the SOFT1 key is pressed, the input mode can be changed. The input mode is in charge of executing the notifykeyInput method of InputMethodHandler that takes care of character handling for the input of number keys. It also has control over the character editing using the full display.

Handling of the arrow key is implemented in the relevant child class, TextFieldComponent, and TextBoxComponent.

**Overrides**

keyNotify in class Component

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released
key	Character of the key pressed; default values are numbers 0 to 9 and symbols * and # (other characters are also possible)

**Return Value**

True when the keys passed on as arguments are handled by this component; otherwise, False is returned

**Class TextFieldComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
    |
    +--org.kwis.msp.lwc.TextComponent
        |
        +--org.kwis.msp.lwc.TextFieldComponent

```

public class **TextFieldComponent** extends TextComponent

This is a class that inherited a TextComponent. A TextFieldComponent edits characters in a line. The width of this component is changed automatically to fit the character data inputted.

TextFieldComponent can edit characters using the full display and can limit inputs to specified strings (refer to the constraint described in TextComponent).

Basically, there is no limit to the size of an input string, although the number of allowable input characters can be limited using TextComponent.setMaxLength(int maxLen).

- **Reference Item**

TextComponent, TextBoxComponent

**Fields inherited from class org.kwis.msp.lwc.TextComponent**

charCount, constChecker, constraint, CONSTRAINT\_ANY, CONSTRAINT\_NUMBER, CONSTRAINT\_PASSWORD, display, f, imHandler, iMode, isWide, m\_cPos, m\_td, maxLength, modeViewer, tShell

**Fields inherited from class org.kwis.msp.lwc.Component**

bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.TextComponent**

focusNotify, getConstraint, getFont, getMaxLength, getString, setFont, setMaxLength, setString, showNotify

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, toString, validate

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **TextFieldComponent**

***public TextFieldComponent(String data, int constraint)***

This method generates an instance of TextFieldComponent using a given character data and constraints. The character data can have a null value.

When a constraint other than TextComponent.CONSTRAINT\_NUMBER, TextComponent.CONSTRAINT\_PASSWORD, or TextComponent.CONSTRAINT\_ANY is specified, IllegalArgumentException is issued. On the other hand, when character data includes data that does not comply with the constraint, IllegalArgumentException is issued.

Basically, there is no limit to the size of an input string, although the number of allowable input characters can be limited using TextComponent.setMaxLength(int maxLen).

**Parameters**

data	haracter data; may also be a null value
constraint	Character input type

### Detailed Description of the Method

- **insert**

***public void insert(char[] data, int offset, int len, int index)***

The following explanation of this method is copied from the **TextComponent** class.

This method adds the character data that is given as an argument in the character data printed on the current display to the index position.

When the data is null, **NullPointerException** is issued. In case of an index value that is smaller than 0 or larger than the total length of the character data that is printed on the current display, **IndexOutOfBoundsException** is issued.

On the other hand, when there is a limit to the number of allowable input characters, and if the aggregate of the length of the character data currently printed and len is larger than the maximum number of allowable input characters, **IllegalArgumentException** is issued.

#### Overrides

Insert in class **TextComponent**

Following copied from class: **org.kwis.msp.lwc.TextComponent**

#### Parameters

data	Character data to be added
offset	Starting position of the character data to be added
len	Length of the character data to be added
index	Position where new character data is to be added in the character data printed on the current display

#### Throws

<b>NullPointerException</b>	Issued when the data is null
<b>IndexOutOfBoundsException</b>	Issued in case of an index value that is smaller than 0 or larger than the total length of the character data printed on the current display
<b>IllegalArgumentException</b>	Issued when there is a limit to the number of allowable input characters, and if the aggregate of the length of the character data currently printed and len is larger than the maximum number of allowable input characters
<b>IllegalArgumentException</b>	Issued when the character data to be inputted includes character data that does not comply with the current constraint

- **delete**

***public void delete(int index, int len)***

The following explanation of this method is copied from the **TextComponent** class.

Data as long as len from the index position of the current character data shown on the display is deleted.

In case of an index value that is smaller than 0 or larger than the total length of the character data printed on the current display, **IndexOutOfBoundsException** is issued. When len exceeds the total length of the character data currently printed, **IllegalArgumentException** is issued.

**Overrides**

Delete in class **TextComponent**

Following copied from class: **org.kwis.msp.lwc.TextComponent**

**Parameters**

index    Position where the data is to be deleted

len      Length of the character data to be deleted

**Throws**

**IndexOutOfBoundsException**    Issued in case of len or index value that is smaller than 0 or larger than the total length of the character data printed on the current display

**IllegalArgumentException**      Issued when len exceeds the total length of the character data currently printed

- **getPreferredWidth**

***public int getPreferredWidth()***

The following explanation of this method is copied from the **Component** class.

This method determines the preferred width of the component. The value returned by this method is referred to when **Container** determines the size of the component.

**Overrides**

getPreferredWidth in class **Component**

Following copied from class: **org.kwis.msp.lwc.Component**

**Return Value**

Width of the component



- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This method determines the preferred width of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight(int wr)***

The following explanation of this method is copied from the Component class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like LabelComponent, TextFieldComponent, or TextAreaComponent, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A w of -1 means that there is no limit on the width.

**Overrides**

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

w      Variable width

**Return Value**

Height of the component

- **paintContent**

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

**Overrides**

paintContent in class Component

Following copied from class: org.kwis.msp.lwc.Component

**Parameters**

g Graphics for painting

**Reference Item**

Graphics

- **keyNotify**

***public boolean keyNotify(int type, int key)***

The following explanation of this method is copied from the TextComponent class.

This method is called when a key input is received. When the user types a key, keyNotify of the component that is given input focus by the setFocus method is called. The type is KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, or KEY\_TYPED, whereas the input value is ch.

(EventQueue) When the SOFT1 key is pressed, the input mode can be changed. The input mode is in charge of executing the notifykeyInput method of InputMethodHandler that takes care of character handling for the input of number keys. It also has control over the character editing using the full display.

**Overrides**

keyNotify in class TextComponent

Following copied from class: org.kwis.msp.lwc.TextComponent

**Parameters**

type	Type of key input, i.e., KEY_PRESSED when a key is pressed; KEY_RELEASED when a key is released; KEY_REPEATED when a key is pressed and held; and KEY_TYPED when a key is pressed once and released.
key	Character of the key pressed; defaults are numbers 0 to 9 and

symbols \* and # (other characters are also possible)

**Return Value**

True when the keys passed over as arguments are handled by this component;  
otherwise, False is returned.

**Class TickerComponent**

```

java.lang.Object
|
+--org.kwis.msp.lwc.Component
|
+--org.kwis.msp.lwc.TickerComponent

```

```

public class TickerComponent extends Component

```

A component that consists of strings and images, TickerComponent moves from right to left. When the end of a string disappears to the left, the start of the string appears again from the right to keep moving in the same direction.

The width value of TickerComponent is the same as that of ContainerComponent that added TickerComponent. For the height, when the image is shorter than the one-line height of the character, it will have a one-line height value. Similarly, when the image is taller than the one-line height of the character, it will have an image height value.

The moving speed can be changed using the setDelay(int) method. Time value will be specified in millisecond. The default value is DEFAULT\_DELAY. A negative (-) value cannot be specified for moving speed. When a negative value is specified, IllegalArgumentException is issued.

TickerComponent provides a method that can control movement. The moving state can be controlled using the setTickerState(boolean) method. The ticker moves when True is specified; otherwise, it stops moving when False is returned. The default value specified for the moving state is True.

TickerComponent cannot be shared by different ContainerComponents. When it is shared, IllegalArgumentException is issued.

**Fields inherited from class org.kwis.msp.lwc.Component**

Bg, evtListener, evtListenerObj, fg, FOCUS\_NOTIFY, h, HAS\_FOCUS\_MASK, INPUT\_MASK, KEY\_NOTIFY, KEY\_PRESSED, KEY\_RELEASED, KEY\_REPEATED, KEY\_TYPED, LAYOUT\_BOTTOM, LAYOUT\_HCENTER, LAYOUT\_LEFT, LAYOUT\_RIGHT, LAYOUT\_TOP, LAYOUT\_VCENTER, mask, parent, POINT\_DRAGGED, POINT\_PRESSED, POINT\_RELEASED, POINTER\_NOTIFY, POS\_MASK, PREFER\_SIZE\_MASK, prefH, prefW, SHOW\_NOTIFY, SIZE\_MASK, VALID\_MASK, w, x, y

**Methods inherited from class org.kwis.msp.lwc.Component**

calcPreferredSize, canHandleInput, configure, focusNotify, getBackground, getCard, getForeground, getHeight, getWidth, getX, getXOnScreen, getY, getYOnScreen, hasFocus, invalidate, isShown, isValid, keyNotify, layout, pointerNotify, processEvent, repaint, repaint, serviceRepaints, setBackground, setEventListener, setFocus, setForeground, showNotify,

toString, validate
--------------------

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
---

#### Detailed Description of the Field

- **DEFAULT\_DELAY**

***public int DEFAULT\_DELAY***

The default moving speed value for TickerComponent is 500 milliseconds.

#### Detailed Description of the Generator

- **TickerComponent**

***public TickerComponent(String str, Image img)***

This method generates an instance of TickerComponent. The default value specified for the moving state is True. In this case, the data is in the moving state. The value for moving speed is DEFAULT\_DELAY.

Character data cannot have a null value. When it is null, NullPointerException is issued. In contrast, image data can have a null value.

##### Parameters

str	Character data of TickerComponent
img	Image data of TickerComponent; may also be a null value

##### Throws

NullPointerException Issued when the character data is null

#### Detailed Description of the Method

- **setString**

***public void setString(String str)***

This method sets the character data of TickerComponent to be printed on the display. In case of data that is already set, the string will be changed to the new data that is set. Character data cannot be null. When a null value is specified for character data, NullPointerException is issued.

##### Parameters

str	Character data of TickerComponent
-----	-----------------------------------

##### Throws

NullPointerException Issued when character data str is null

##### Reference Item

getString(), setImage(Image img)

- **getString**

***public String getString()***

Gets the character data of TickerComponent

**Return Value**

Character data of the string TickerComponent

**Reference Item**

setString(String str)

- **setImage**

***public void setImage(Image img)***

This method sets the image data of TickerComponent. Any data that is already set will be changed to the newly set data.

**Parameters**

img     Image data of TickerComponent; when img is null, the existing image is deleted

**Reference Item**

getImage(), setString(String str)

- **getImage**

***public Image getImage()***

Transmits the image data of TickerComponent

**Return Value**

Image data of the img TickerComponent

**Reference Item**

setImage(Image img)

- **setDelay**

***public void setDelay(int delay)***

This method sets the character flow speed of TickerComponent. The default value for flow speed is DEFAULT\_DELAY.

The larger the flow speed is, the slower the character flow speed will be. This value is in

millisecond, and a negative value cannot be set. When a negative value is set, `IllegalArgumentException` is issued.

**Parameters**

delay    Character speed value in millisecond

**Throws**

`IllegalArgumentException`    Issued when delay is a negative value

- **setTickerState**

***public boolean setTickerState(boolean st)***

This method sets the moving/stop state for `TickerComponent`.

When `True` is specified, `TickerComponent` moves; otherwise, when `False` is returned, it stops. The default value for the moving state is `True`.

**Parameters**

st    Moving characters and images when `True`; when `False`, movement stops

- **getPreferredHeight**

***public int getPreferredHeight(int w)***

The following explanation of this method is copied from the `Component` class.

This method determines the preferred height of the component. The height of the component is returned in case the component has a specified, limited width. In case of a component that can be formatted like `LabelComponent`, `TextFieldComponent`, or `TextAreaComponent`, the component can have a variable width. In this case, the height varies depending on the width. The height can also be obtained using this method. A `w` of `-1` means that there is no limit on the width.

**Overrides**

`getPreferredHeight` in class `Component`

Following copied from class: `org.kwis.msp.lwc.Component`

**Parameters**

w    Variable width

**Return Value**

Height of the component

- **getPreferredHeight**

***public int getPreferredHeight()***

The following explanation of this method is copied from the Component class.

This is a method that determines the preferred height of the component. The value returned by this method is referred to when ContainerComponent determines the size of the component.

#### Overrides

getPreferredHeight in class Component

Following copied from class: org.kwis.msp.lwc.Component

#### Return Value

Height of the component

#### ● getPreferredWidth

***public int getPreferredWidth()***

The following explanation of this method is copied from the Component class.

This method determines the preferred width of the component. The value returned by this method is referred to when Container determines the size of the component.

#### Overrides

getPreferredWidth in class Component

Following copied from class: org.kwis.msp.lwc.Component

#### Return Value

Width of the component

#### ● paintContent

***public void paintContent(Graphics g)***

The following explanation of this method is copied from the Component class.

This method paints the interior. The validate method is first called to validate (i.e., recalculation of the component's position and size) the position of the component. The display is then painted with an internal color. Painting is not carried out when the color is -1.

#### Overrides

paintContent in class Component

Following copied from class: org.kwis.msp.lwc.Component

#### Parameters

g Graphics for painting

#### Reference Item



Graphics

### 3.1.5. Generic I/O

#### Class IODevice

```

java.lang.Object
|
+---org.kwis.msp.io.IODevice

public class IODevice extends java.lang.Object

```

Defines a class that is used to control generic I/O devices

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Generator

- IODevice

***public IODevice(java.lang.String devname, int devnum, byte[] param) throws IOException***

The name and number of devices supported can be obtained by transmitting "supported.iodevices" as a parameter of the System.getProperty() method. In case of more than two identical I/O devices, they are distinguished by devnum that is transmitted as a parameter. For example, when there are two IrDAs, the first device becomes No. 0, and the second device, No. 1. The name and parameter data of a device are defined as shown in the table below (the name and parameter that are not defined in the table should comply with the Generic I/O of C API and specifications of the MC\_ioDevOpen() method):

Devname	Param
"IrDA"	1st byte: Sets the mode, i.e., server mode as 'S', client mode as 'C' 2nd~5th bytes: sets Timeout; saved in big-ending format of byte array
"1ChipCard"	Creates byte array objects for parameters; after calling this method, get the ATR (Answer To Reset) value

#### Parameters

devname	Device name
devnum	Device number
param	arameter to be passed on when opening the device

**Throws**

IOException      Issued when device opening fails

**Detailed Description of the Method**● **close**

***public void close() throws IOException***

Closes an open IODevice

**Throws:**

IOException      Issued in case of an error

● **read**

***public int read(byte[] buf, int offset, int length) throws IOException***

Reads data from an IODevice (a (blocking) method)

**Parameters**

buf	storage space for data to be read
offset	start offset
length	length of data to be read

**Return Value**

Number of actually read bytes

**Throws:**

IOException      Issued in case of an error

● **write**

***public int write(byte[] buf, int offset, int length) throws IOException***

Writes data in an IODevice (a (blocking) method)

**Parameters**

buf	data to be written
offset	start offset
length	length of data to be written

**Return Value**

Number of actually written bytes

**Throws:**

IOException      Issued in case of an error

- **control**

***public void control(java.lang.String cmd, byte[] param1, byte[] param2) throws IOException***

This method controls an IODevice and carries out an operation according to a command given to a device. The formats for cmd, param1, and param2 are defined as shown in the table below (others that are not defined in the table should follow the General I/O Specifications of C API):

Device	cmd	Param
"IrDA"	"SETOPCODE" (Sets the transmission method for IrDA)	Param1: [in] "GETMETHOD" Param2: [in] "OBEXGET" or "OBEXPUT"
"1ChipCard"	"GETSTATUS" (Checks whether or not the IC Card is inserted) "GETCHANNEL" (Gets the logical channel no. of currently assigned UICC)	Param1: [in] byte array of length 7 [out] "exist" or "noexist" Param2: [in] byte array of length 4 [out] integer-type channel no. with big-ending format

**Parameters**

cmd	ring representing the type of operation to be carried out by a device
param1, param2	Parameters to be passed on to the relevant operation of a device

**Return Value**

None

### 3.1.6. Terminal Resource

#### Class ResourceGroup

java.lang.Object

|

+---org.kwis.msp.io.ResourceGroup

public class **ResourceGroup** extends java.lang.Object

Defines a class related to a terminal resource group

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Generator

- **ResourceGroup**

***public ResourceGroup(java.lang.String grpName) throws IOException***

This method generates a ResourceGroup object that fits a group name (grpName). The resource group currently supported by the terminal can be determined when the supports.resourcegroups value is obtained using the System.getProperty() method.

#### Parameters

grpName      ResourceGroup name to be generated

#### Detailed Description of the Field

- **GROUP\_LOCKED**

***public static final int GROUP\_LOCKED***

Has a value of 2

- **GROUP\_UNLOCKED**

***public static final int GROUP\_UNLOCKED***

Has a value of 3

- **LOCKED**

***public static final int LOCKED***

Has a value of 0

- **UNLOCKED**

***public static final int UNLOCKED***

Has a value of 1

### Detailed Description of the Method

- **checkPassword**

***public static boolean checkPassword( java.lang.String password)***

When the relevant resource is locked, checks whether a password given to release the lock matches the password that is required to unlock it

**Parameters**

password	Password
----------	----------

**Return Value**

True when successful; otherwise, in case of invalid password, false is returned

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

- **deleteData**

***public void deleteData(String resName)***

Deletes a resource with a corresponding name

**Parameters**

resName	Resource name
---------	---------------

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
IOException	Issued when there is no access authorization (M_E_ACCESSDENY) or no delete method (M_ENOTSUP) or in case of disabled delete method (M_E)NODELETE)

- **getCount**

***public int getCount()***

Returns the number of resources stored in ResourceGroup

**Return Value**

Number of resources when successful; otherwise, an error code (a negative

value) is returned

- **getData**

***public byte[] getData(java.lang.String resName)***

Returns the actual data of a resource corresponding to the resource name

**Parameters**

resName      Resource name

**Return Value**

Resource data

**Throws**

IllegalArgumentException      Issued when the transferred parameter is invalid (in case of C API, the same case as M\_E\_INVALID error)

- **getFreeSpace**

***public int getFreeSpace()***

Gets the size of free space for the resource group storage space in byte

**Return Value**

Free space for the resource group storage space

- **getFormat**

***public java.lang.String getFormat(java.lang.String resName)***

Returns the resource format corresponding to the resource name of ResourceGroup

**Parameters**

resName      Resource name

**Return Value**

Returns the MIME type as a string

**Throws**

IllegalArgumentException      Issued when the transferred parameter is invalid (in case of C API, the same case as M\_E\_INVALID error)



- **getList**

***public java.lang.String[] getList()***

Returns a resource name list for ResourceGroup

**Return Value**

Resource name list

- **getGroupLockStatus**

***public int getGroupLockStatus() throws IOException***

Returns the lock status for the group resource (lock status includes GROUP\_LOCKED and GROUP\_UNLOCKED)

**Parameters**

None

**Return Value**

Lock status

**Throws**

IOException	Issued when the corresponding resource group (or resource) does not provide the LOCK method (M_E_NOTSUPPORTLOCK, M_E_NOT_SUPPORTGLOCK)
-------------	--

- **getLockStatus**

***public int getLockStatus(java.lang.String resName) throws IOException***

Returns the lock status of the resource corresponding to a given resource name (lock status includes LOCKED and UNLOCKED)

**Parameters**

resName	Resource name
---------	---------------

**Return Value**

Lock status

**Throws**

IOException	Issued when the corresponding resource group (or resource) does not provide the LOCK method (M_E_NOTSUPPORTLOCK, M_E_NOT_SUPPORTGLOCK)
-------------	--

- **getRegisteredInfo**

***public static java.lang.String[] getRegisteredInfo(java.lang.String state) throws IOException***

Returns a list of resource names registered in a specific terminal state from among resources belonging to a resource group

**Parameters**

state	Terminal state
"IDLE"	Idle screen
"INCOMING"	Incoming screen
"POWERON"	Power on screen
"POWEROFF"	Power off screen
"BROWSERON"	Browser on state
"BROWSEROFF"	Browser off state

**Return Value**

List of resource names and resource groups composed in the sequence of resource group name, ",", and resource name

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
IOException	Issued in case of other error

- **getSize**

***public int getSize(java.lang.String resName)***

Returns the actual size of data for a resource corresponding to a resource name

**Parameters**

resName	Resource name
---------	---------------

**Return Value**

Actual size of data for a resource

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

- **getSupportedGroups**

***public static java.lang.String[] getSupportedGroups()***

This method returns a list of resource groups that are currently supported by the terminal. This method has the same method as `Syste.getProperty ("supported.resourcegroups")`, provided that the value to be returned is a list of strings.

**Return Value**

A list of ResourceGroup names provided

- **registerData**

***public void registerData(java.lang.String resName, java.lang.String state)***  
***throws IOException***

Sets a resource corresponding to a resource name in a specific terminal state

**Parameters**

resName	resource name
state	terminal state
"IDLE"	idle screen
"INCOMING"	incoming screen
"POWERON"	power on screen
"POWEROFF"	power off screen
"BROWSERON"	Browser on state
"BROWSEROFF"	Browser off state

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
IOException	Issued in case of other error (no access authorization, register method unsupported)

- **setGroupLockStatus**

***public void setGroupLockStatus(int status)***

Sets or releases the lock on a resource corresponding to a group

**Parameters**

Status Lock status (GROUP\_LOCKED, GROUP\_UNLOCKED)

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

- **setLockStatus**

***public void setLockStatus(java.lang.String resName, int status) throws***  
***IOException***

Sets or releases the lock on a resource corresponding to a given name

**Parameters**

resName      Resource name  
status   Lock status (LOCKED, UNLOCKED)

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
IOException	Issued when the corresponding resource group (or resource) does not provide the LOCK method (M_E_NOTSUPPORTLOCK, M_E_NOTSUPPORTGLOCK)

- **writeData**

***public java.lang.String writeData (java.lang.String title, java.lang.String uiName, java.lang.String format, byte[] data, boolean update) throws IOException***

Writes resource data with a specific format

**Parameters**

title	Resource name that is either new or already existing
uiName	Name displayed on UI
format	MIME type of the resource
data	Data
update	Update resource with resource name

**Return Value**

Returns the resource name stored as a title; when it is treated as an error, a null value is returned

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
IOException	Issued when there is no writing authorization or when the writing method is not supported

- **exists**

***public boolean exists(java.lang.String resName)***

Verifies if a resource corresponding to a specified resource name exists

**Parameters**

resName	Resource name
---------	---------------

**Return Value**

Whether or not the resource exists (True when the resource exists; otherwise, False is returned)

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

- **getGroupInfo**

***public byte[] getGroupInfo(java.lang.String type)***

This method gets the characteristics of a group corresponding to type for a resource group. The characteristics and type of a group that can be obtained for each group should follow the definition of the MC\_termResGetGroupInfo() method of C API.

**Parameters**

type      Type information for obtaining the characteristics of a group

**Return Value**

Information on characteristics corresponding to a group  
type; null in the absence of information on characteristics

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

- **getInfo**

***public byte[] getInfo(java.lang.String resName, java.lang.String type)***

This method gets the characteristics of a resource corresponding to type for a resource. The resource characteristics and type that can be obtained for each resource should follow the definition of the MC\_termResGetInfo() method of C API.

**Parameters**

resName	Resource name
type	Type information for obtaining the characteristics of a group

**Return Value:**

Information on characteristics corresponding to a resource  
type; null in the absence of information on characteristics

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

- **search**

***static public String[] search(java.lang.String grp, java.lang.String type, java.lang.String query, boolean exactMatch)***

This method searches a resource that matches a string search word given according to a resource group and search word type. A list of resource names is returned as the search result. The specifications method of search type should follow the definition of the MC\_termResSearch() method of C API.

**Parameters**

grp	resource group name
type	search type
query	search word
exactMatch	Search method (when True is returned, only resources that completely match the search word are searched)

**Return Value:**

A list of search results;  
null in the absence of the search results

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---

**● getUIName**

***public java.lang.String getUIName(java.lang.String resName)***

***Returns the UI name of the specified resource***

**Parameters**

resName	Resource name
---------	---------------

**Return Value**

UI name

**Throws**

IllegalArgumentException	Issued when the transferred parameter is invalid (in case of C API, the same case as M_E_INVALID error)
--------------------------	---



### 3.1.7. Address Book

#### Class Address

```
java.lang.Object
|
+--org.kwis.msp.handset.Address
```

public static class **Address** extends java.lang.Object

A class representing an individual address record of an address book

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Method

- **getRecordId**

***public int getRecordId()***

Record ID of an address object

#### Return Value

Record ID of an address object

- **setShortcut**

***public boolean setShortcut(int shortcut, int fieldIndex)***

This method can be used only in case of only 1 shortcut supported in each record. It sets the shortcut and key number. In case of only 1 shortcut supported in each record, the record ID and the shortcut should match. Therefore, the successful execution of this method causes the record ID to change. If a record is already assigned to the shortcut to be changed, this method is not executed. This method cannot delete a shortcut.

#### Parameters

shortcut	Number of shortcuts to be set
fieldIndex	Key number field index; in case of -1, current key number is reserved

#### Return Value

true	successfully changed the field value
false	changing the field value failed

- **getField**

***public Object getField(int fieldIndex)***

Gets the field value of an address object

**Parameters**

fieldIndex      Index of an address field

**Return Value**

Field object of an address object

- **setField**

***public boolean setField(int fieldIndex, Object field)***

Sets the field

**Parameters**

fieldIndex      Index of an address field  
field

**Return Value**

**true**

Changes the field value

**false**

Field value unchanged

- **getFields**

***public Object[] getFields()***

Gets multiple fields

**Return Value**

Array of field objects

- **setFields**

***public boolean setFields(Object[] fields)***

Sets multiple fields

**Parameters**

fields      Array of field objects

**Return Value****true**

Changes the field value

**false**

Field value unchanged

- **getLockStatus**

***public int getLockStatus()***

Returns the lock status of an address object (lock status includes LOCKED and UNLOCKED)

**Return Value**

Lock status

- **setLockStatus**

***public int setLockStatus(int status)***

Sets/releases the lock on an address object

**Parameters**

status Lock status (LOCKED, UNLOCKED)

**Return Value**

0 when successful; otherwise, an error code (a negative value) is returned

**Class AddressBook**

java.lang.Object

|

+--org.kwis.msp.handset.**AddressBook**public static class **AddressBook** extends java.lang.Object

A class that is used to access an address book of a terminal

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field**● **SEARCH\_NAME***public static final int SEARCH\_NAME*

Used when the search reference field is a name

● **SEARCH\_PHONE\_NO***public static final int SEARCH\_PHONE\_NO*

Used when the search reference field is a phone number

● **SEARCH\_EMAIL***public static final int SEARCH\_EMAIL*

Used when the search reference field is an e-mail address

● **SEARCH\_GROUP***public static final int SEARCH\_GROUP*

Used when the search reference field is a group

● **TYPE\_INT***public static final int TYPE\_INT*

Field type; representing the field as a number

● **TYPE\_STRING***public static final int TYPE\_STRING*

Field type; representing the field as a string

● **TYPE\_IMAGE***public static final int TYPE\_IMAGE*

Field type; representing the field as an image data

- **TYPE\_SOUND**

***public static final int TYPE\_SOUND***

Field type; representing the field as an audio clip

- **TYPE\_BINARY**

***public static final int TYPE\_BINARY***

Field type; representing the field as a binary data

- **LOCKED**

***public static int LOCKED***

Indicates LOCKED status

- **UNLOCKED**

***public static int UNLOCKED***

Indicates UNLOCKED state

#### Detailed Description of the Method

- **getGroupCount**

***public int getGroupCount()***

Gets the number of available groups (categories) in an address book

**Return Value**

Number of available groups (categories) in an address book

- **getGroupName**

***public String getGroupName(int groupId)***

Gets a group (category) name

**Parameters**

GroupId          Index of the group

**Return Value**

Group name

- **createGroup**

***public int createGroup(String groupName)***

Creates a new group (category)

**Parameters**

groupName                      Group name

**Return Value**

Group index

- **getAddressBook**

***static public AddressBook getAddressBook()***

Gets an instance of a global AddressBook

**Return Value**

Global AddressBook object

- **getFieldCount**

***public int getFieldCount()***

This method gets the number of fields constituting a record. The field index can be determined through the field count ( 0 ~ (getFieldCount()-1)).

**Return Value**

Field count

- **getFieldName**

***public String getFieldName(int fieldIndex)***

Gets the name of each field

**Parameters**

fieldIndex                      fieldIndex ( 0 ~ (getFieldCount()-1)), i.e.,  
   gets a name

**Return Value**

Field name

- **getFieldType**

***public int getFieldType(int fieldIndex)***

Gets the field type of each field

**Parameters**

fieldIndex                      fieldIndex ( 0 ~ (getFieldCount()-1))

**Return Value**

Data type, TYPE\_INT, TYPE\_STRING, ...

- **getFieldMaxLength**

***public int getFieldMaxLength(int fieldIndex)***

Gets the data length of each field

**Parameters**

fieldIndex      fieldIndex ( 0 ~ (getFieldCount()-1))

**Return Value**

Field length

- **getAddressMaxCount**

***public int getAddressMaxCount()***

Gets the maximum number of records that can be created in an address book

**Return Value**

Maximum number of records that can be created in an address book

- **getAddressCount**

***public int getAddressCount()***

Gets the number of records being used in an address book

**Return Value**

Number of records being used in an address book

- **getAddressRecordIdsAll**

***public int[] getAddressRecordIdsAll()***

Gets the IDs of all records being used

**Return Value**

IDs of all records being used

- **getAddress**

***public Address getAddress(int recordId)***

Gets an address object

**Parameters**

recordId      Record ID

**Return Value**

Address object

- **createRecord**

***public int createRecord(Object[] fields)***

Creates a record

**Parameters**

fields    Field data of a record to be created

**Return Value**

ID of a record created

- **createRecords**

***public int[] createRecords(Object[] records)***

Creates multiple records

**Parameters**

records    Array of record data; each record data is an array of field data

**Return Value**

Array of record IDs created

- **isSupportFieldShortCut**

***public boolean isSupportFieldShortCut()***

Distinguishes whether a shortcut is created for each record or field

**Return Value**

**true**

Supported shortcut for each field

**false**

Unsupported shortcut for each field

- **isSupportShortCut**

***public boolean isSupportShortCut(int fieldIndex)***

API that is supported only when shortcuts are supported; checks whether a field supports a shortcut



**Parameters**

fieldIndex      Field index

**Return Value**

**true**

A field supporting a shortcut

**false**

A field that does not support a shortcut

- **getMaxShortCut**

***public int getMaxShortCut()***

Gets the maximum number of shortcuts that can be set

**Return Value**

Maximum number of shortcuts that can be set

- **getFirstFreeShortCut**

***public int getFirstFreeShortCut(int startShortCut)***

API that is supported only when shortcuts by field are supported; gets the smallest value from among all available shortcuts that are larger than or the same size as startShortCut

**Parameters**

startShortCut    Shortcut number where search is to start

**Return Value**

Smallest value from among all available shortcuts that are larger than or similar to startShortCut; when there is no shortcut, -1 is returned

- **setShortCut**

***public boolean setShortCut(int shortCut, int recordId, int fieldId)***

API that is supported only when shortcuts by API field are supported; specifies a shortcut to a specific field of a specific record (new record and field cannot be assigned a shortcut to which a record and a field are already assigned; to assign a new record and a field, free a shortcut by assigning "recordId = -1" to the relevant shortcut before assigning setShortCut() again)

**Parameters**

shortCut	Shortcut number to be set
recordId	Target record ID for which a shortcut is to be set
fieldId	Target Field ID for which a shortcut is to be set

**Return Value**

**True**

Shortcut that is set

**False**

Shortcut that is not set

- **setShortCut**

***public boolean setShortCut(int[] shortCut, int[] recordId, int[] fieldId)***

API that is supported only when shortcuts by field are supported; sets a shortcut to a specific field of a specific record

**Parameters**

shortCut	int array containing a list of shortcuts
recordId	int array containing a list of record IDs
fieldId	int array containing a list of field indexes

**Return Value**

**True**

Shortcut that is set

**False**

Shortcut that is not set

- **getAllShortCut**

***public int[] getAllShortCut()***

API that is supported only when shortcuts by field are supported; gets a list of all shortcuts that are assigned

**Return Value**

A list of all shortcuts that are assigned

- **getShortCutItem**

***public int[] getShortCutItem(int shortCut)***

Determines a record ID and a Field ID that are assigned a shortcut

**Parameters**

shortCut	Shortcut that is assigned to a record and a field to be queried
----------	---

**Return Value**

int[0] = record Id int[1]=field Index

- **getShortCutAssigned**

***public int getShortCutAssigned(int recordId, int fieldIndex)***

Checks whether a shortcut is assigned to a specific field of a specific record

**Parameters**

recordId	Record ID
fieldIndex	Field index

**Return Value**

ShortCut number -1 when a shortcut is not assigned

- **searchAddress**

***public int[] searchAddress(int searchBy, Object field, boolean exactMatch)***

Searches a record that satisfies a given condition

**Parameters**

searchBy	Name (SEARCH_NAME), telephone number (SEARCH_PHONE_NO), e-mail (SEARCH_EMAIL), and group (SEARCH_GROUP)
field	Data to be found (integer type when searching a current group; otherwise, it is string type)
exactMatch	When True, searches a record that exactly matches the field data; when False, searches a record with a field containing the field data

**Return Value**

ID of a record determined

- **removeAddress**

***public boolean removeAddress(int recordId)***

Removes a specific address record

**Parameters**

recordId	ID of an address object to be removed
----------	---------------------------------------

**Return Value**

**true**

Address removed

**false**

Address not removed

- **checkPassword**

***public int checkPassword(int index, java.lang.String password)***

When the resource corresponding to a specific index (dwIndex) is locked, checks whether a password for releasing the lock matches the password given

**Parameters**

index	Resource index
password	Password

**Return Value**

0 when successful; otherwise, an error code (a negative value) is returned

- **getLockStatus**

***public int getLockStatus()***

Returns the lock status of an address object (lock status includes LOCKED and UNLOCKED)

**Return Value**

Returns the lock status

- **setLockStatus**

***public int setLockStatus(int status)***

Sets/releases the lock of an address book object

**Parameters**

status Lock status (LOCKED, UNLOCKED)

**Return Value**

0 when successful; otherwise, an error code (a negative value) is returned

### 3.1.8. Media Handler

#### Class BaseClip

```
java.lang.Object
|
+--org.kwis.msp.media.BaseClip
```

public abstract class **BaseClip** extends java.lang.Object

An uppermost abstract class used to implement media handler

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Method

- **allocPlayer**

***protected int allocPlayer()***

This method allocates a player used to handle media data in a clip. It should be called before actual handling such as play can be implemented using media data in a clip. Otherwise, if a method such as play is called without calling this method, or if this method fails, an error is returned.

**Parameters**

None

**Return Value**

0 when successful; otherwise, a negative value is returned

- **freePlayer()**

***protected int freePlayer()***

Releases the player allocated using the allocPlayer() method

**Parameters**

None

**Return Value**

0 when successful; otherwise, a negative value is returned

- **free**

***public void free()***

All the resource used within the clip are disabled. Calling this method causes the clip data or resources used to be removed from the memory.

**Parameters**

None

**Return Value**

None

- **mediaWriteData**

***protected int mediaWriteData()***

This method copies media data in a clip to the inner buffer of the media handler. Data as much as the maximum volume of the clip inner buffer are copied, and actual written volume, returned.

**Parameters**

None

**Return Value**

Written size

- **mediaReadData**

***protected int mediaReadData()***

This method copies data in the media handler to the buffer in a clip. Data as much as the maximum volume of the clip inner buffer are read, and actual read volume, returned.

**Parameters**

None

**Return Value**

Read size

- **mediaControl**

***protected static int mediaControl(int cmd, int[] buf1, int[] buf2);***

This method calls the MH\_mdaControl() method of HAL. The meaning of the transmitted value and return value matches the MH\_mdaControl() method.

**Parameters**

(refer to the MH\_mdaControl() method)

**Return Value**

(refer to the MH\_mdaControl() method)

- **mediaControl**

***protected static int mediaControl(int cmd, int[] buf1, byte[] buf2);***

This method calls the MH\_mdaControl() method of HAL. The meaning of the transmitted value and return value matches the MH\_mdaControl() method.

**Parameters**

(refer to the MH\_mdaControl() method)

**Return Value**

(refer to the MH\_mdaControl() method)

- **mediaControl**

***protected static int mediaControl(int cmd, String buf1, int[] buf2);***

This method calls the MH\_mdaControl() method of HAL.

The meaning of the transmitted value and return value matches the MH\_mdaControl() method.

**Parameters**

(refer to the MH\_mdaControl() method)

**Return Value**

(refer to the MH\_mdaControl() method)

- **mediaModeControl**

***public int mediaModeControl(String modeName, int cmd, int pID, int[] buf);***

This method calls the MH\_mdaModeControl() method of HAL. The meaning of the transmitted value and return value matches the MH\_mdaControl() method.

**Parameters**

(refer to the MH\_mdaModeControl() method)

**Return Value**

(refer to the MH\_mdaModeControl() method)



- **mediaModeControl**

***public int mediaModeControl(String modeName, int cmd, int pID, byte[] buf);***

This function calls the MH\_mdaModeControl() method of HAL. The meaning of the passed value and return value matches the MH\_mdaControl() function.

**Parameters**

(refer to the MH\_mdaModeControl() method)

**Return Value**

(refer to the MH\_mdaModeControl() method)

- **mediaModeControl**

***public int mediaModeControl(String modeName, int cmd, int pID, String buf);***

This function calls the MH\_mdaModeControl() method of HAL. The meaning of the passed value and return value matches the MH\_mdaControl() function.

**Parameters**

(refer to the MH\_mdaModeControl() method)

**Return Value**

(refer to the MH\_mdaModeControl() method)

- **mediaDeviceControl**

***protected static int mediaDeviceControl(int cmd, int[] buf1, int[] buf2);***

This method calls the MH\_mdaDevControl() method of HAL. The meaning of the transmitted value and return value matches the MH\_mdaDevControl() method.

**Parameters**

(refer to the MH\_mdaDevControl() method)

**Return Value**

(refer to the MH\_mdaDevControl() method)

- **mediaDeviceControl**

***protected int mediaDeviceControl(int cmd, int[] buf1, byte[] buf2);***

This function calls the MH\_mdaDevControl() method of HAL. The meaning of the passed value and return value matches the MH\_mdaDevControl() function.

**Parameters**

(refer to the MH\_mdaDevControl() method)

**Return Value**

(refer to the MH\_mdaDevControl() method)

- **mediaDeviceControl**

***protected int mediaDeviceControl( int cmd, String buf1, int[] buf2);***

This function calls the MH\_mdaDevControl() method of HAL. The meaning of the passed value and return value matches the MH\_mdaDevControl() function.

**Parameters**

(refer to the MH\_mdaDevControl() method)

**Return Value**

(refer to the MH\_mdaDevControl() method)

- **medialInfo**

***public static int medialInfo()***

This method calls the MH\_mdaGetDeviceInfo() method of HAL. The return value matches the meaning of the value returned with rtnInfo in MH\_mdaGetDeviceInfo (M\_Int32 devID, M\_Int32\* rtnInfo).

**Parameters**

(refer to MH\_mdaGetDeviceInfo())

**Return Value**

(refer to MH\_mdaGetDeviceInfo())

- **setWaterMark**

***public void setWaterMark(int percent)***

This method sets the WaterMark below which the PlayListner.END\_OF\_DATA and PlayListner.FULL\_OF\_DATA events are to be generated. In case the WaterMark is set to 90%, and more than 90% of the data stored in a clip is played, the PlayListner.END\_OF\_DATA event occurs. If more than 90% of the storage buffer in a clip is filled while recording, the PlayListner.FULL\_OF\_DATA event occurs. When an event occurs given the WaterMark set, the event occurs only once, and the WaterMark setting is automatically disabled. In other words, when the WaterMark is set to 90% for playing, and an event occurs once when more than 90% of the data is played, the event will not recur even when 93% or 95% of the data is played. To generate the event again, the reactiveWaterMark() method is called. The default is 0%.

**Parameters**

[in]     percent     WaterMark (0-100)

**Return Value**

None

**Throws**

IllegalArgumentException	Issued in case of invalid percent value
--------------------------	---

- **reactiveWaterMark**

***public void reactiveWaterMark()***

This method enables a disabled WaterMark setting. If the WaterMark setting is already enabled, this method does not play any role. An event set to be activated when the WaterMark is exceeded will occur when this method is implemented, and the conditions are satisfied.

**Parameters**

None

**Return Value**

None

- **setBuffer**

***public boolean setBuffer(byte[] buf, int dataSize)***

This method sets the internal buffer of a clip. The buf transmitted as a parameter will be used as the internal buffer of a BaseClip. Otherwise, when a buffer is not created when creating BaseClip, this method is used to set an internal buffer after a BaseClip object is created.

**Parameters**

buf	Buffer
-----	--------

dataSize	Size of data contained in the buffer
----------	--------------------------------------

**Return Value****Pass**

true

**Fail**

false	Buffer already set
-------	--------------------

- **putData**

***public int putData(byte[] buf, int off, int len)***

This method copies media data to a clip. Media data should have a data type that is set at the time when the clip is created. Data in a clip is reduced when it is played in a media player and can be increased by putData(byte[] buf, int off, int len). When the size of the data value to be copied is larger than the data to be accommodated by the internal buffer of a clip, only the size of the data value that can be accommodated is copied.

**Parameters**

buf	Data buffer
off	Buffer offset
len	Size to be copied

**Return Value**

Size copied

**Throws**

IndexOutOfBoundsException	Issued in case of invalid values for off and len
NullPointerException	Issued when buf is null

- **getData**

***public int getData(byte[] buf, int off, int len)***

This method copies media data from a clip to a buffer. The size of the data in a clip increases when it is recorded in a media player. It can be reduced by getData(byte[] buf, int off, int len). When data in a clip is larger than the buffer transmitted, only the same volume as the buffer size is copied.

**Parameters**

buf	Buffer where data in a clip is to be copied
off	Start position of the copying
len	Size to be copied

**Return Value**

Size copied

**Throws**

IndexOutOfBoundsException	Issued in case of invalid values for off and len
NullPointerException	Issued when buf is null

- **clearData**

***public void clearData()***

All available data in a clip cleared

**Parameters**

None

**Return Value**

None

- **availableDataSize**

***public int availableDataSize()***

Data size available in a clip (not the size of an internal buffer of a clip)

**Parameters**

None

**Return Value**

Size of available data

- **playStart**

***protected boolean playStart (boolean repeat)***

This method calls the repeat values as parameters before calling the actual play method in the Player.play(Clip clip, boolean repeat) method. Any work to be done before the actual media play is read should be set here.

**Parameters**

repeat Repeat values transmitted to Player.play()

**Return Value**

**true**

Player.play() method normally implemented

**false**

Player.play() method no longer implemented; returned as False

- **recordStart**

***protected boolean recordStart()***

This method is called before the actual record method in the Player.record(BaseClip clip)

method is called. Any work to be done before the actual media record is read should be set here.

**Parameters**

None

**Return Value**

**true**

Player.record() method normally implemented

**false**

Player.record () method no longer implemented; returned as False

- **playUpdate**

***public boolean playUpdate(int event, int param)***

Indicates the status change when playing a clip (the event to be transmitted is the same as `PlayListener.playUpdate()`)

**Parameters**

event Status value

param Used in case of an additional value to be transmitted to each event

**Class Clip**

```

java.lang.Object
|
+--org.kwis.msp.media.BaseClip
   +--org.kwis.msp.media.Clip

```

public class **Clip** extends Clip

Class implementing a clip that is to be played by a player

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field****Detailed Description of the Generator**

- **Clip**

***public Clip(String type, byte[] buf)***

This method receives parameters that already store data and creates a clip.

**Parameters**

type	Resource type
buf	Buffer containing data

**Throws**

IllegalArgumentException	Issued when trying to create a type that isn't supported
--------------------------	--

- **Clip**

***public Clip(String type)***

This method creates a clip of a specific type without creating an inner buffer. When creating and subsequently setting a buffer, `setBuffer()` is used.

**Parameters**

type	Resource type
------	---------------

**Throws**

IllegalArgumentException	Issued when trying to create an unsupported type
--------------------------	--

- **Clip**

***public Clip(String type, int bufSize)***

This method creates a clip of a specific type in a specified size. The specific type is one of the types obtained by "MEDIADEVICES" of `HandsetProperty.getProperty()`. Data input is executed by inputting data to be played into the CLIP buffer through `putData(byte[] buf, int off, int len)`. Since the capability of the media player varies depending on the device, the size of the CLIP inner buffer should be set to continuous play when playing the media.

**Parameters**

type	Resource type
bufSize	Buffer size

**Throws**

<code>IllegalArgumentException</code>	Issued when trying to create an unsupported type
---------------------------------------	--

- **Clip**

***public Clip(String type, String resourceName)***

This method receives resource names that already store data and creates a clip. It is the same as `Clip(String type, byte[] buf)`.

**Parameters**

type	Resource type
resourceName	Resource name containing data

**Throws**

<code>IllegalArgumentException</code>	Issued when trying to create an unsupported type
---------------------------------------	--



**Detailed Description of the Method**● **getType*****public java.lang.String getType()***

Gets the type of a clip

**Return Value**

Type string

● **setPosition*****public boolean setPosition(int ms)***

This method sets the starting point of playing in milliseconds. When calling this method to a clip of an unsupported type, `MediaUnsupportedException` is issued.

**Parameters**

ms      Starting point to start a clip play (millisecond)

**Return Value****true**

Position set

**false**

Position not set

**Throws**`MediaUnsupportedException`

Issued when calling this method to a clip of an unsupported type

● **getPosition*****public boolean getPosition()***

This method gets the current playing time in milliseconds. When calling this method to a clip of an unsupported type, `MediaUnsupportedException` is issued.

**Parameters**

None

**Return Value**

Current playing time (in milliseconds)

**Throws**

MediaUnsupportedException	Issued when calling this method to a clip of an unsupported type
---------------------------	--

- **setStopTime**

***public boolean setStopTime(int ms)***

This method sets the stop point of playing in milliseconds. When calling this method to a clip of an unsupported type, MediaUnsupportedException is issued. To release the set stop point, input -1 into the parameter ms.

**Parameters**

ms	Starting point to stop a clip play (in milliseconds)
----	--

**Return Value**

**true**

Success

**false**

Failure

**Throws**

MediaUnsupportedException	Issued when calling this method to a clip of an unsupported type
---------------------------	--

- **getStopTime**

***public int getStopTime()***

This method gets the set stop point in milliseconds. When calling this method to a clip of an unsupported type, MediaUnsupportedException is issued.

**Parameters**

None

**Return Value**

Set stop point (in milliseconds)

**Throws**

MediaUnsupportedException	Issued when calling this method to a clip of an unsupported type
---------------------------	--

- **getVolume**

***public final int getVolume()***

This method reads the volume from a clip player. When the independent volume setting of a clip player is supported, this method reads the volume from a clip player. Otherwise, when the independent volume setting is unsupported, this method can indicate the same volume source even if the clip creation type is different. The minimum value for the volume is 0, whereas the maximum value is 100.

**Parameters**

None

**Return Value**

**Pass**

Volume value (0 - 100)

- **setVolume**

***public final boolean setVolume(int level)***

This method sets the volume for a clip player. When the independent volume setting of a clip player is supported, this method sets the volume for a clip player. Otherwise, when the independent volume setting is unsupported, this method can indicate the same volume source even if the clip creation type is different. The minimum volume value to be set is 0, whereas the maximum value is 100. In case of invalid level, it is ignored.

**Parameters**

level    Volume value (0 - 100)

- **setListener**

***public void setListener(PlayListener listener)***

Registers a listener that will indicate the status change when playing a clip

**Parameters**

Listener            New listener; when null, the existing listener is removed

**Class Player**

java.lang.Object

|

+---org.kwis.msp.media.**Player**public class **Player** extends java.lang.Object

A class that includes the necessary static method for playing a media

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Method**● **pause*****public static boolean pause(BaseClip clip)***

This method stops media handling (play/recording) temporarily. When this method is called, and media handling is stopped temporarily, a state of PAUSE is transmitted to the event listener method registered in the clip. When called again for a handler that is paused or stopped, this method does not play any role. On the other hand, when pause is attempted on a clip created with an unsupported type, MediaUnsupportedException is issued.

**Parameters**

clip      Clip to be stopped temporarily

**Return Value****true**

Pass

**false**

Already paused or stopped

● **stop*****public static boolean stop(BaseClip clip)***

This method stops media handling (play/recording). When this method is called, and media handling is stopped, a state of STOP is transmitted to the event listener method registered in the clip. When called again for a handler that is stopped, this method does not play any role.

**Parameters**

clip     Clip to be stopped

**Return Value**

**true**

Pass

**false**

Already stopped

- **resume**

***public static boolean resume(BaseClip clip)***

This method resumes the media handling (play/recording) that is temporarily stopped. When this method is called, and media handling is resumed, a state of RESUME is transmitted to the event listener method registered in the clip. When called again for a handler that is handling media, this method does not play any role. On the other hand, when resume is attempted on a clip created with a type that does not support resume, `MediaUnsupportedException` is issued.

**Parameters**

clip     Clip to be resumed

**Return Value**

**true**

Resumed

**false**

Currently handling media

- **play**

***public static boolean play(BaseClip clip, boolean repeat)***

This method plays data in a clip. When this method is called, and media handling starts, a state of START is transmitted to the event listener method registered in the clip. In case the playing of the transmitted clip is not possible because another clip is being played, `UnavailableException` is issued. This method does not play any role when replay is attempted even with the clip being played. When all clip data is consumed, an

END\_OF\_MEDIA state is transmitted to the event listener method. Likewise, when streaming play is preferred, an END\_OF\_DATA state is transmitted to the event listener method before the clip data is completely used (when getting the water mark). In case of remaining data to be played at this time, clip data should be filled using the Clip.putData() method.

**Parameters**

clip	Clip to be played
repeat	One time play when False; repeat play when True

**Return Value**

**true**

Pass

**false**

Play in progress

- **record**

***public static boolean record(BaseClip clip)***

This method starts recording. When attempting to record with a clip created with a type that does not support recording, MediaUnsupportedException is issued. On the other hand, when this method is called, and media handling starts, a state of RECORD is transmitted to the event listener method registered in the clip.

When recording is not possible because another clip is being recorded, UnavailableException is issued. This method does not play any role when rerecording is attempted even with a clip being recorded. When the internal buffer of a clip is completely filled while recording, a END\_OF\_MEDIA state is transmitted to the event listener method. Likewise, when recording is executed by streaming recording, the FULL\_OF\_DATA state is transmitted to the event listener method before the internal buffer of a clip is completely filled (when getting the water mark). To continue recording at this time, the internal buffer of the clip should be emptied using the Clip.GetData() method.

**Parameters**

clip	Clip where recording data is to be stored
------	---

**Return Value**

**true**

Pass

**false**

Recording in progress

**Interface PlayListener**

public interface **PlayListener**

Used in an application program that wants to determine the status change of a media player

**Detailed Description of the Field**

- **ERROR**

*public static final int ERROR*

Value of -1 for error occurrence

- **END\_OF\_DATA**

*public static final int END\_OF\_DATA*

Value of 1 for arriving at the end of play data

- **START**

*public static final int START*

Value of 2 for play that is started

- **STOP**

*public static final int STOP*

Value of 3 for play/recording that is stopped

- **PAUSE**

*public static final int PAUSE*

Value of 4 for play/recording that is paused

- **RESUME**

*public static final int RESUME*

Value of 5 for the resumption of temporarily stopped play/recording

- **RECORD**

*public static final int RECORD*

Value of 6 for recording that is started

- **FULL\_OF\_DATA**

*public static final int FULL\_OF\_DATA*

Value for recording buffer completely filled; recording cannot proceed any longer (7)

- **END OF MEDIA**

*public static final int END\_OF\_MEDIA*

All media data have been played completely (8).



- **STOP AT TIME**

***public static final int STOP\_AT\_TIME***

The play stopped at the stop point set using the setStopTime method (9).

Detailed Description of the Method

- **playerUpdate**

*public void playerUpdate(BaseClip clip, int event, int param)*

Called in case of a status change while playing a clip

**Parameters**

Clip	Clip where a status change occurred
event	Status value
param	Used in case of an additional value to be transmitted to each event

### Class MediaUnavailableException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +-- org.kwis.msp.media.MediaUnavailableException
  
```

public class UnavailableException extends **RuntimeException**

An exception class that occurs when a resource is not available

#### Methods inherited from class java.lang.Throwable

getMessage, printStackTrace, toString

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

#### Detailed Description of the Generator

- **MediaUnavailableException**

***public MediaUnavailableException()***

Generates an MediaUnavailableException

- **MediaUnavailableException**

***public MediaUnavailableException(String s)***

Generates an MediaUnavailableException

#### Parameters

s                      Detailed message of MediaUnavailableException

**Class Vibrator**

```

java.lang.Object
|
+--org.kwis.msp.media.Vibrator

```

public class **Vibrator** extends java.lang.Object

A class that controls the vibration of a handset

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Method**

- **on**

***public static void on(int level, int duration)***

After this method is returned, a vibrator should vibrate for a specified period of time. If the vibrator is already vibrating before this method is called, the vibration should be initialized and carried out for a specified period of time using the Parameters duration. The Timeout value is valid only when the value for the Parameters level is larger than 0. A level value of 0 means turning off a vibrator. Vibration intensity is determined by the values of the Parameters level, which may be between 0 and 100. 100 represents the strongest vibration supported by the hardware, whereas 0 represents the weakest vibration. Matching the values of 0-100 with the level of vibration strength depends on how vibration levels supported by the hardware match the percentage as shown below (ignored in case of invalid level and duration).

The levels of vibration supported by the hardware can be determined through `HandsetProperty.getProperty("VIBRATORLEVEL")`.

Ex.) Hardware with one vibration strength => 1-100: Vibration

Hardware with two vibration strengths => 1-50: weak vibration; 51-100: strong vibration

Hardware with three vibration strengths => 1-33: weak vibration; 34-66: Intermediate vibration; 67-100: strong vibration

**Parameters**

level	When 0, vibration is off; when 1-100, vibration strength is matched by the OS
duration	Time the vibration lasts (millisecond); when 0, the duration of vibration is unlimited

- **off**

***public static void off()***

This method ends the vibration (ignored in case of vibration that already ended).

**Class Volume**

```
java.lang.Object
|
+--org.kwis.msp.media.Volume
```

```
public class Volume extends java.lang.Object
```

A collection of methods used to adjust the sound volume

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Field**

- **VOLTYPE\_VOICE**

```
public static final int VOLTYPE_VOICE
```

Denotes the volume of voice; the value is 1

- **VOLTYPE\_RING**

```
public static final int VOLTYPE_RING
```

Denotes the volume of ring; the value is 2

- **VOLTYPE\_KEYTONE**

```
public static final int VOLTYPE_KEYTONE
```

Denotes the volume of key tone; the value is 3

- **VOLTYPE\_MESSAGE**

```
public static final int VOLTYPE_MESSAGE
```

Denotes the volume of SMS message; the value is 4

- **VOLTYPE\_ALARM**

```
public static final int VOLTYPE_ALARM
```

Denotes the volume of alarm; the value is 5

- **VOLTYPE\_ALERT**

```
public static final int VOLTYPE_ALERT
```

Denotes the volume of alert; the value is 6

- **VOLTYPE\_MMEDIA**

***public static final int VOLTYPE\_MMEDIA***

Denotes the master volume of all multimedia devices; the value is 7

#### Detailed Description of the Method

- **get**

***public static int get()***

This method returns the value of volume. When an independent volume is set for each device, this value may not be accurate. In such cases, the accurate volume of each device should be read using `Clip.getVolume()`.

The volume value to be returned should be converted to a value between 0 and 100. Matching the values of 0-100 with the level of vibration strength depends on how vibration levels supported by the hardware are matched with the percentage as shown below. The volume strength to be supported by the hardware is returned using `HandsetProperty.getSystemProperty("VOLUMELEVEL")`.

Ex.) Hardware with two vibration strengths => 1-50: weak vibration; 51-100:

strong vibration

Hardware with three vibration strengths => 1-33: weak vibration; 34-66:

intermediate vibration; 67-100: strong vibration

**Parameters**

None

**Return Value**

Volume value

- **set**

***public static void set(int level)***

This method sets the volume for all devices for which the volume can be set. When the independent volume is set for each device, volume API in a clip can be used. The minimum value for the volume to be set is 0, whereas the maximum value is 100 (ignored in case of invalid level).

**Parameters**

level    Volume value (0-100)

● **setMute**

***public static void setMute(int volType, boolean mute) throw  
IllegalArgumentException***

This method sets the mute state for a terminal by volume type based on the value of volType. Included in the VolType values are VOLTYPE\_VOICE, VOLTYPE\_RING, VOLTYPE\_KEYTONE, VOLTYPE\_MESSAGE, VOLTYPE\_ALARM, VOLTYPE\_ALERT, and VOLTYPE\_MMEDIA. When a value other than these values is transmitted as volType, IllegalArgumentException is issued.

**Parameters**

[in]	volType	VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.	
[in]	mute	Sets the mute state	
		TRUE	Sets the mute state
		FALSE	Releases the mute state

**Throws**

IllegalArgumentException	Issued not in case of volType other than VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.
--------------------------	---



- **getMute**

***public static boolean getMute(int volType) throw IllegalArgumentException***

This method gets the mute state for a terminal by volume type based on the value of volType. Included in the VolType values are VOLTYPE\_VOICE, VOLTYPE\_RING, VOLTYPE\_KEYTONE, VOLTYPE\_MESSAGE, VOLTYPE\_ALARM, VOLTYPE\_ALERT, and VOLTYPE\_MMEDIA. When a value other than these values is transmitted as volType, IllegalArgumentException is issued.

**Parameters**

[in]	volType	VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA .
------	---------	--

**Return Value**

**TRUE**

Mute obtained

**FALSE**

Mute not obtained

**Throws**

IllegalArgumentException	Issued not in case of a volType other than VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.
--------------------------	---

- **setDefaultVolume**

***public static void setDefaultVolume(int volType, int vol)***

***throw IllegalArgumentException***

This method sets the default volume for a terminal by volume type based on the value of volType. Included in the VolType values are VOLTYPE\_GENERAL, VOLTYPE\_VOICE, VOLTYPE\_RING, VOLTYPE\_KEYTONE, VOLTYPE\_MESSAGE, VOLTYPE\_ALARM, VOLTYPE\_ALERT, and VOLTYPE\_MMEDIA. When a value other than these values is transmitted as volType, IllegalArgumentException is issued.

**Parameters**

[in]	volType	VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.
[in]	vol	Volume value (0-100)

**Throws**

IllegalArgumentException	Issued not in case of a volType other than VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.
--------------------------	---

- **getDefaultVolume**

***public static int getDefaultVolume(int volType) throw IllegalArgumentException***

This method gets the default volume for a terminal by volume type based on the value of volType. Included in the volType values are VOLTYPE\_GENERAL, VOLTYPE\_VOICE, VOLTYPE\_RING, VOLTYPE\_KEYTONE, VOLTYPE\_MESSAGE, VOLTYPE\_ALARM, VOLTYPE\_ALERT, and VOLTYPE\_MMEDIA. When a value other than these values is transmitted as volType, IllegalArgumentException is issued.

**Parameters**

[in]	volType	VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.
------	---------	---

**Return Value**

Volume value

**Throws**

IllegalArgumentException	Issued not in case of a volType other than VOLTYPE_VOICE, VOLTYPE_RING, VOLTYPE_KEYTONE, VOLTYPE_MESSAGE, VOLTYPE_ALARM, VOLTYPE_ALERT, or VOLTYPE_MMEDIA.
--------------------------	---

**Class MediaUnsupportedException**

```

java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--org.kwis.msp.media.MediaUnsupportedException

```

public class **MediaUnsupportedException** extends Exception

An exception class issued when an attempt to use an unsupported method is made

**Methods inherited from class java.lang.Throwable**

getMessage, printStackTrace, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **MediaUnsupportedException**

***public MediaUnsupportedException()***

Generates MediaUnsupportedException

- **MediaUnsupportedException**

***public MediaUnsupportedException(String s)***

Generates MediaUnsupportedException

**Parameters**

s      Detailed message of IOException

## Camera

```

java.lang.Object
|
+--org.kwis.msp.media.BaseClip
      |
      +--Camera
          |
          public class Camera extends BaseClip
  
```

Defines the camera class used to control a camera device as follows:

### Detailed Description of the Field

- **DETECT**

***protected static final int DETECT***

Camera detection control command; the value is 0

- **MODEL**

***protected static final int MODEL***

Control command that gets the name of a model; the value is 1

- **GET\_MODE\_LIST**

***protected static final int GET\_MODE\_LIST***

Control command that gets a list of mode names; the value is 2

- **SET\_MODE**

***protected static final int SET\_MODE***

Control command that sets a mode; the value is 3

- **SET\_AXIS**

***protected static final int SET\_AXIS***

Control command that rotates/reverses the display; the value is 4

- **PREVIEW\_START**

***protected static final int PREVIEW\_START***

Control command that starts a preview; the value is 5

- **PREVIEW\_STOP**

***protected static final int PREVIEW\_STOP***

Control command that stops a preview; the value is 6

- **CAPTRUE\_INTERVAL**

***protected static final int CAPTRUE\_INTERVAL***

Control command that sets the time interval between frames when playing or storing

videos; the value is 7

- **NORMAL**

***public static final int NORMAL***

Normal display; the value is 0

- **HORZ\_REVERSE**

***public static final int HORZ\_REVERSE***

Horizontal reverse; the value is 1

- **VERT\_REVERSE**

***public static final int VERT\_REVERSE***

Horizontal reverse; the value is 2

- **BOTH\_REVERSE**

***public static final int BOTH\_REVERSE***

Both horizontal and vertical reverses; the value is 3

- **ROTATE90**

***public static final int ROTATE90***

90° rotation to the right; the value is 4

- **ROTATE180**

***public static final int ROTATE180***

180° rotation to the right; the value is 5

- **ROTATE270**

***public static final int ROTATE270***

180° rotation to the right; the value is 6

## Detailed Description of the Generator

- **Camera**

***public Camera(String type)***

This method generates a clip that uses a camera device, and the result is returned. The parameter `Type` is transmitted in mime type supported by the camera device, "video/mpeg" when the data recorded by the camera device is mpeg type, and "video/h.263" when the h.263 format is supported.

### Parameters

type    Resource type

- **Camera**

***public Camera(String type, int bufSize)***

This method generates a clip that uses a camera device, and the result is returned. The parameter Type is transmitted in mime type supported by the camera device, "video/mpeg" when the data recorded by the camera device is mpeg type, and "video/h.263" when the h.263 format is supported.

**Parameters**

type     Resource type  
bufSize   Size of buffer containing data

**Detailed Description of the Method**

- **init**

***public synchronized boolean init()***

This method initializes the camera particularly the camera sensor and the controller chip and turns on the power. It assigns a buffer for storing images in HAL.

**Return Value**

**true**

Camera initialized

**false**

Camera not initialized

- **close**

***public synchronized void close()***

This method ends the camera. It turns off the power of the camera sensor and the controller chip and erases the buffer for storing images in HAL.

**Return Value**

None

- **detect**

***public static boolean detect()***

Detects whether or not a camera is equipped

**Return Value**



**true**

Camera detected

**false**

Camera not detected

- **getModel**

***public String getModel()***

Gets a camera model name

**Return Value**

Camera model name

- **setMode**

***public boolean setMode(int mode)***

This method sets a camera mode for play/recording on a clip. The mode includes picture quality, resolution, and display position on LCD. Information on a mode may differ depending on a phone based on the porting method of HAL.

**Parameters**

mode	Mode number support starting from 0; when four modes are supported, mode numbers are from 0 to 3 (0 always indicates the default mode)
------	--

**Return Value**

**true**

Mode set

**false**

Mode not set

- **getModeCount**

***public int getModeCount()***

Gets the number of modes supported

**Return Value**

Number of modes (when the number is 3, modes supported are 0, 1, and 2)

- **setProperty**

***public boolean setProperty(int property)***

Sets the property for play/recording

**Parameters**

property	Properties including NORMAL, HORZ_REVERSE, VERT_REVERSE, BOTH_REVERSE, ROTATE90, ROTATE180, and ROTATE270
----------	---

**Return Value**

**true**

Property set

**false**

Property not set

- **setSize**

***public boolean setSize(int x, int y, int width, int height)***

This method specifies the customized OEM display area for a currently set mode. When each coordinate is out of the LCD display area, the values for the MAX LCD display area are applied.

**Parameters**

x	X coordinate of the customized area
y	Y coordinate of the customized area
width	Width of the customized area
height	Height of the customized area

**Return Value**

**true**

Size set

**false**

Unsupported specifications of the customized area

- **disableOEMDisplayArea**

***public void disableOEMDisplayArea()***

Disables the OEM display area defined in each mode

- **enableOEMDisplayArea**

***public void enableOEMDisplayArea()***

This method enables the OEM display area (camera display) defined in each mode. In case this method is not called, the area displayed by the camera and the area displayed by the platform can overlap. When this method is displayed, only the camera display area can be displayed regardless of what is performed on the platform. In contrast, when the platform is to draw something in the camera display area, the disableOEMDisplayArea() method should be called first.

- **previewStart**

***public void previewStart()***

Starts camera preview

- **previewStop**

***public void previewStop()***

Stops camera preview

**StillClip**

```

java.lang.Object
|
+--org.kwis.msp.media.BaseClip
|
+--org.kwis.msp.media.Camera
|
+--org.kwis.msp.media.StillClip

```

Defines a StillClip class that controls a camera device to capture and play still images as follows:

**Detailed Description of the Generator**

- **StillClip(String type)**

***public StillClip(String type)***

This method generates a still image clip. The parameter Type is transmitted with mime type supported in the camera device. When the stored image data when taking pictures using the camera device is jpeg type, it becomes "image/jpeg," and "image/png" when the png format is supported.

**Parameters**

type      Resource type

- **StillClip(String type, int bufSize)**

***public StillClip (String type, int bufSize)***

This method generates a still image clip whose buffer size in the clip is bufSize. The parameter Type is transmitted with mime type supported in the camera device. When the stored image data when taking pictures using the camera device is jpeg type, it becomes "image/jpeg," and "image/png" when the png format is supported.

**Parameters**

type                      Resource type  
bufSize                  Size of buffer containing data

**Detailed Description of the Method**

- **snapshot**

***public boolean snapshot(PlayListener listener)***

This method takes still images using a camera. When this method is called, the camera device starts shooting in background and copies the taken images to the buffer in a clip. To determine the time when the taken images are completely copied to the buffer in the

clip, the listener is transmitted as a parameter. When the taken images are completely copied to the buffer in the clip, FULL\_OF\_DATA is transmitted to the playUpdate() method of the listener. Before shooting starts, however, the buffer in the clip should be cleared using the clearData() method.

**Parameters**

listener Listener

**Return Value**

**true**

Shooting started

**false**

Shooting not started

- **view**

***public boolean view(PlayListener listener)***

This method prints the content stored in the buffer on the display. When this method is called, the camera device starts playing in the background and copies the taken images from the buffer in a clip to the camera device. To determine the time when the taken images are completely played in the camera device, the listener is transmitted as a parameter. When the taken images are completely played in the media handler, FULL\_OF\_DATA is transmitted to the playUpdate() method of the listener. After playing, the data is still stored in the buffer of the clip.

**Parameters**

listener Listener

**Return Value**

**true**

Play started

**false**

Play not started

- **getData**

***public byte[] getData()***

This method gets the stored data from the buffer. After this method is called, the data stored in the buffer is removed.

**Return Value**

byte array	Data obtained
null	No data stored in the buffer of the clip

- **playStart**

***protected boolean playStart (boolean repeat)***

This method calls the repeat values as parameters before calling the actual play method in the `Player.play(Clip clip, boolean repeat)` method. Any work to be done before the actual media play occurs should be set here.

**Parameters**

repeat	Repeat values transmitted to <code>Player.play()</code>
--------	---

**Return Value**

**true**

`Player.play()` method normally implemented

**false**

`Player.play()` method no longer implemented; returned as False

- **playUpdate**

***public boolean playUpdate(int event, int param)***

The following explanation of this method is copied from the **Clip Class**.

This method indicates the status change while playing a clip. The event transmitted is the same as `PlayListener.playUpdate()`.

**Overrides**

`playUpdate` in class `Clip`

Following copied from class: `org.kwis.msp.media.Clip`

**Parameters**

event	Status value
param	Used in case of an additional value to be transmitted to each event

**VideoClip**

```

java.lang.Object
|
+--org.kwis.msp.media.BaseClip
    |
    +--org.kwis.msp.media.Camera
        |
        +--org.kwis.msp.media.VideoClip
  
```

Defines a VideoClip class that controls a camera device to record and play video images as follows:

**Detailed Description of the Generator**

- **VideoClip(String type)**

***public VideoClip(String type)***

This method generates a video clip. The parameter Type is transmitted with mime type supported in the camera device. When the recorded data in the camera device is mpeg type, it becomes "video/mpeg," and "video/h.263" when the h.263 format is supported.

**Parameters**

type    Resource type

- **VideoClip(String type, int bufSize)**

***public VideoClip(String type, int bufSize)***

This method generates a video clip whose buffer size in the clip is bufSize. The parameter Type is transmitted with mime type supported in the camera device. When the recorded data in the camera device is mpeg type, it becomes "video/mpeg," and "video/h.263" when the h.263 format is supported.

**Parameters**

type    Resource type

bufSize Size of buffer containing data

**Detailed Description of the Method**

- **record**

***public boolean record(PlayListener listener)***

When this method is called, recording starts in background, and the data taken in a camera device is copied to the buffer in a clip. When the buffer of the clip is completely filled, the data taken thereafter are discarded. In case stop () is not called until the buffer of the clip is completely filled, FULL\_OF\_DATA is transmitted to the playUpdate() method of the listener transmitted as a parameter.

**Return Value****true**

Recording started

**false**

Recording not started

● **pause*****public boolean pause()***

Stops recording/play temporarily

**Return Value****true**

Recording/play stopped temporarily

**false**

Recording/play not stopped temporarily

● **resume*****public boolean resume()***

Resumes recording/play that is temporarily stopped

**Return Value****true**

Recording/play resumed

**false**

Recording/play not resumed

● **stop*****public boolean stop()***

Stops recording/play

**Return Value****true**



Recording/play stopped

**false**

Recording/play not stopped

- **play**

***public boolean play(PlayListener listener)***

When this method is called, playing starts in background, and the data in the buffer of a clip is copied to a camera device. When the buffer of the clip is completely emptied, the display of play screen is thereafter dependent on a phone. When stop () is not called until data in the buffer of the camera is completely played, END\_OF\_DATA is transmitted to the playUpdate () method of the listener transmitted as a parameter. After playing, the data is still stored in the buffer of the clip.

**Parameters**

listener                  Listener

**Return Value**

**true**

Play started

**false**

Play not started

- **playStart**

***protected boolean playStart (boolean repeat)***

This method calls repeat values as parameters before calling the actual play method in the Player.play(Clip clip, boolean repeat) method. Any work to be done before the actual media play is read should be set here.

**Parameters**

repeat                  Repeat values transmitted to Player.play()

**Return Value**

**true**

Player.play() method normally implemented

**false**

Player.play() method no longer implemented; returned as False

- **playUpdate**

***public boolean playUpdate(int event, int param)***

The following explanation of this method is copied from the Clip Class.

This method indicates the status change when playing a clip. The event to be transmitted is the same as PlayListener.playUpdate().

**Overrides**

playUpdate in class Clip

Following copied from class: org.kwis.msp.media.Clip

**Parameters**

event	Status value
param	Used in case of an additional value to be transmitted to each event

### 3.1.9. SMS

#### Class SMSMessage

```

java.lang.Object
|
+--org.kwis.msp.io.SMSMessage

public class SMSMessage extends java.lang.Object

```

Provides Short Message Service (SMS) of a terminal

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Generator

- **SMSMessage**

***public SMSMessage(Byte[] data)***

Generates the SHORT\_MESSAGE type of SMS messages that can be sent using the SMS.send() method

**Parameters**

data[] Message data

#### Detailed Description of the Field

- **SHORT\_MESSAGE**

***public static final int SHORT\_MESSAGE***

Defines a short character message of a terminal

- **UNKNOWN**

***public static final int UNKNOWN***

Displays constant information on an unknown character message on a terminal

**Class SMS**

java.lang.Object

|

+--org.kwis.msp.io.**SMS**public class **SMS** extends java.lang.Object

Provides Short Message Service of a terminal together with SMSMessage

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

None

**Detailed Description of the Field**● **M\_E\_NOTSUP*****public static final int M\_E\_NOTSUP***

An error that occurs when the relevant teleservice is unsupported

● **M\_E\_INVALID*****public static final int M\_E\_INVALID***

An error that occurs when the size of a buffer is invalid

● **M\_E\_WOULDLOCK*****public static final int M\_E\_WOULDLOCK***

An error that occurs when data cannot be transmitted immediately due to an internal factor of the system

● **M\_E\_ERROR*****public static final int M\_E\_ERROR***

Other errors

## Detailed Description of the Method

- **send**

***public int send(String telnum, SMSMessage smsMsg)***

This method sends SMSMessage. When the send() method is called, a verification window for the transmission of SMS messages is displayed. SMSMessage is sent only when the user chooses "OK" on the verification window.

**Parameters**

telNum	Telephone number of the party being contacted
smsMsg	SMSMessage object

**Return Value**

0 when the message is transmitted normally; otherwise, an error is returned as follows:

M_E_NOTSUP	When the relevant teleservice is unsupported
M_E_INVALID	When the size of the buffer is invalid
M_E_WOULDBLOCK	When data cannot be transmitted immediately due or of the system
M_E_ERROR	Other errors

**Throws**

NullPointerException Issued when telNum or smsMsg is null

- **getMaxMsgLength**

***public static int getMaxMsgLength()***

Converts the length of a message that can be transmitted to byte before it is returned

**Parameters**

None

**Return Value**

**Pass**

Maximum length of an SMS message in byte

**Fail**

M_E_ERROR	Request failed (cause: unknown)
-----------	---------------------------------

**Side Effect**

None

**Reference Item**

None

### 3.1.10. Location Information

#### Class StationLocationInfo

java.lang.Object

|

+--org.kwis.msp.handset. **StationLocationInfo**

public class **StationLocationInfo** extends java.lang.Object

Provides location information based on the station method

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Generator

None

#### Detailed Description of the Method

- **getBaseID**

***public int getBaseID()***

Reads the Station ID

**Return Value**

Station ID

- **getBaseLatitude**

***public int getBaseLatitude()***

Reads the station latitude

**Return Value**

Station latitude

- **getBaseLongitude**

***public int getBaseLongitude()***

Reads the station longitude

**Return Value**

Station longitude

- **isValid**

***public boolean isValid()***

Reads whether or not the station location information is valid

**Return Value**

Valid when True is returned; invalid when False is returned

- **getLocationInfo**

***public final int getLocationInfo () throws IOException***

This is an API that requests for the location information of a station communicated by terminals. In case of a successful processing of the request, the values requested in baseID, baseLat, and baseLong fields of the class are stored, and stationInfoValid is set to True.

**Parameters**

None

**Return Value**

O      issued when the information is obtained; otherwise, an exception is issued

**Throws**

IOException      Issued when the location information cannot be read



**Class GPSTConfig**

java.lang.Object

|

+---org.kwis.msp.handset.**GPSTConfig**public class **GPSTConfig** extends java.lang.Object

Provides location information based on GPS

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**● **GPSTConfig**

***public GPSTConfig(int mode, int qos, int transport, string pde\_addr, int pde\_port) throws IllegalArgumentException***

Generates a GPSTConfig class used to initialize a GPS device using the given GPS configuration

**Parameters**

mode	gpsOne™ location information reception mode
qos	gpsOne™ quality level
transport	gpsOne™ information transmission layer
pdeAddr	PDE server address
pdePort	PDE server port

**Throws**

IllegalArgumentException      Issued when the given configuration value is invalid

**Detailed Description of the Field**● **MS\_ASSISTED**

***public static final int MS\_ASSISTED***

Sets the location information reception mode to MS\_ASSISTED (the value is 1)

● **MS\_BASED**

***public static final int MS\_BASED***

Sets the location information reception mode to MS\_BASED (the value is 2)

- **OPT SPEED**

***public static final int OPT\_SPEED***

Sets the location information reception mode to speed optimization (the value is 3)

- **OPT ACCURACY**

***public static final int OPT\_ACCURACY***

Sets the location information reception mode to accuracy optimization (the value is 4)

- **SERVER TCPIP**

***public static final int SERVER\_TCPIP***

Sets the information transmission layer to TCPIP (the value is 1)

- **SERVER DBURST**

***public static final int SERVER\_DBURST***

Sets the information transmission layer to DBURST (the value is 2)

#### Detailed Description of the Method

- **getMode**

***public int getMode()***

Reads the gpsOne™ location information reception mode

**Return Value**

gpsOne™ location information reception mode

- **getQos**

***public int getQos()***

Reads the gpsOne™ quality level

**Return Value**

gpsOne™ quality level

- **getTransport**

***public int getTransport()***

Reads the gpsOne™ information transmission layer

**Return Value**

gpsOne™ information transmission layer

- **getPDEAddress**

***public String getPDEAddress()***

Reads the PDE server address

**Return Value**

PDE server address

- **getPDEPort**

***public int getPDEPort()***

Reads the PDE server port

**Return Value**

PDE server port

- **setMode**

***public void setMode(int mode)***

Sets the gpsOne™ location information reception mode; the GPS device is also changed to the set value

**Parameters**

mode    gpsOne™ location information reception mode

**Throws**

IllegalArgumentException	Issued when the set value is invalid
GPSException	Issued when an error related to the GPS device occurs

- **setQos**

***public void setQos(int qos)***

Sets the gpsOne™ quality level; the GPS device is also changed to the set value

**Parameters**

qos    gpsOne™ quality level

**Throws**

IllegalArgumentException	Issued when the set value is invalid
GPSException	Issued in case of an error related to the GPS device

- **setTransport**

***public void setTransport(int transport)***

Sets the gpsOne™ information transmission layer; the GPS device is also changed to the set value

**Parameters**

transport          gpsOne™ information transmission layer

**Throws**

IllegalArgumentException	Issued when the set value is invalid
GPSEException	Issued in case of an error related to the GPS device

- **setPDEAddress**

***public void setPDEAddress (String pdeAddr)***

Sets the gpsOne™ PDE server address; the GPS device is also changed to the set value

**Parameters**

pdeAddr                  PDE server address

**Throws**

IllegalArgumentException	Issued when the set value is invalid
GPSEException	Issued in case of an error related to the GPS Device

- **setPdePort**

***public void setPDEPort (int pdePort)***

Sets the PDE server port; the GPS device is also changed to the set value

**Parameters**

pdePort                  PDE server port

**Return Value**

0 when the operation is successful

**Throws**

IllegalArgumentException	Issued when the set value is invalid
GPSEException	Issued in case of an error related to the GPS device

**Class GPSLocationInfo**

java.lang.Object

|

+--org.kwis.msp.handset.**GPSLocationInfo**public class **GPSLocationInfo** extends java.lang.Object

Provides location information based on GPS

**Methods inherited from class java.lang.Object**

Equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

GPSLocationInfo

public GPSLocationInfo()

Generates a class for GPSLocationInfo

**Detailed Description of the Method**● **getLatitude*****public int getLatitude()***

Reads the latitude value of GPS location information

**Return Value**

Latitude value

● **getLongitude*****public int getLongitude()***

Reads the longitude value of GPS location information

**Return Value**

Longitude value

● **getAltitude*****public int getAltitude()***

Reads the height value of GPS location information

**Return Value**

Height value

- **getHeading**

***public int getHeading ()***

Reads the direction value of GPS location information

**Return Value**

Direction value

- **getHorizontalVelocity**

***public int getHorizontalVelocity()***

Reads the horizontal speed value of GPS location information

**Return Value**

Horizontal speed value

- **getVerticalVelocity**

***public int getVerticalVelocity()***

Reads the vertical speed value of GPS location information

**Return Value**

Vertical speed value

- **getAccuracy**

***public int getAccuracy()***

Reads the accuracy value of GPS location information

**Return Value**

Accuracy value

- **getTimeStamp**

***public String getTimeStamp()***

Reads the time stamp value of GPS location information

**Return Value**

Time stamp string (HHMMSS), e.g., "231050": 23 (hour) 10 (minute) 50 (second)

- **isValid**

***public boolean isValid()***

Indicates whether or not the GPS information (latitude, longitude, altitude, heading, velocityHor, velocityVer, accuracy, timeStamp, etc.) is valid; default value is False (set to True when GPS information is properly acquired)

**Return Value**

Valid when True, invalid when False

### Class GPSPProvider

java.lang.Object

|

+--org.kwis.msp.handset.**GPSPProvider**

public final class **GPSPProvider** implements SystemEventListener

Provides location information based on GPS

### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Detailed Description of the Generator

GPSPProvider

public GPSPProvider()

Generates a class for GPSPProvider

### Detailed Description of the Field

- **REQUEST\_ONCE**

***public static final int REQUEST\_ONCE***

Carries out the GPS location information query once; the value is 0

- **REQUEST\_STOP**

***public static final int REQUEST\_STOP***

Stops the GPS location information repeat query; the value is -1

- **RESULT\_SUCCESS**

***public static final int RESULT\_SUCCESS***

Successfully received GPS location information (the value is 1)

- **RESULT FAILED**

***public static final int RESULT\_FAILED***

Receiving GPS location information failed (the value is 2)

- **RESULT BEGIN**

***public static final int RESULT\_BEGIN***

GPS begins without error on request (the value is 3)

- **RESULT NOT CONNECT**

***public static final int RESULT\_NOT\_CONNECT***

Cannot connect to the PDE server (the value is 4)

- **RESULT INPROGRESS**

***public static final int RESULT\_INPROGRESS***

Already in progress in another application (the value is 5)

- **RESULT LOCKED**

***public static final int RESULT\_LOCKED***

GPS method locked (the value is 6)

### Detailed Description of the Method

- **available**

***public static boolean available() throws GPSEException***

API that inquires whether a GPS device is available

**Parameters**

None

**Return Value**

true when available; otherwise, false is returned

**Throws**

GPSEException	Issued in case of an error related to the GPS device, i.e., when there is no GPS device, or it is being used by another
---------------	---



application

- **requestLocationInfo**

***public static final int requestLocationInfo(int interval)***

This is an API that requests for location information based on gpsOne. It operates asynchronously, and the result is disclosed to an event. When the event is received, locationInfoReceived() of GPSTListener registered by the user is called to obtain the result (success/failure) and the location information.

**Parameters**

interval	Repeat query stops at REQUEST_STOP; requests once at REQUEST_ONCE; reports a number above 1 every interval second (if possible)
----------	---

**Return Value**

0 when the operation is successful

**Throws**

GPSEException	Issued in case of error related to the GPS device, i.e., when there is no GPS device, it is being used by another application, or the parameter is invalid
---------------	--

- **getGPSTConfig**

***public static GPSTConfig getGPSTConfig (void)***

Gets the gpsOne™ configuration information

**Return Value**

GPSTConfig when the operation is successful

**Throws**

GPSEException	Issued in case of an error related to the GPS device
---------------	--

- **setGPSTConfig**

***public static void setGPSTConfig (GPSTConfig config)***

Sets the gpsOne™ configuration information

**Parameters**

configGPSTConfig
------------------

**Throws**

GPSEException	Issued in case of an error related to the GPS device
---------------	--



- **setLocationInfoListener**

***public static boolean setLocationInfoListener (GPSListener listener)***

This method registers a GPSListener. Only one listener can be registered on the platform. When the application registered as a listener is terminated, the listener is released.

**Parameters**

listener GPSListener to be registered

**Return Value**

true in case of successful GPSListener registration; otherwise, false is returned

**Class GPSEException**

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--org.kwis.msp.handset.GPSEException
  
```

public class **GPSEException** extends Exception

Issued in a typical or exceptional situation related to a GPS device

**Methods inherited from class java.lang.Throwable**

getMessage, printStackTrace, toString

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

**Detailed Description of the Generator**

- **GPSEException**

***public GPSEException(String str)***

Generates a new instance together with a message

**Parameters**

message                      Detailed message on the exception

- **GPSEException**

***public GPSEException()***

Generates a new GPSEException instance

**Interface GPSListener**

public interface **GPSListener**

An interface that indicates the occurrence of an event when GPS information has been received

**Detailed Description of the Method**

- **LocationInfoReceived**

***public void LocationInfoReceived(GPSLocationInfo info, int result)***

Called when a GPS event occurs after requesting location information

**Parameters**

info    GPSLocationInfo information (valid only in case of success)

result    Code for success/failure (refer to the constant value of GPSPProvider)

### 3.1.11. Control of Supplementary Devices

#### Class HandsetProperty

```
java.lang.Object
|
+--org.kwis.msp.handset.HandsetProperty
```

public class **HandsetProperty** extends Object

A class that manages the specified values for a terminal

#### Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Detailed Description of the Generator

None

#### Detailed Description of the Method

- **getSystemProperty**

***public static String getSystemProperty (String id) throws  
IllegalArgumentException***

This method reads the specified values for a terminal. An idstring that can be used as a parameter should be modeled after the string used in MH\_sysGetInformation() of HAL document API. It can be added or expanded depending on the communication service provider or a vendor.

##### Parameters

id      Identification string for a specified value

##### Return Value

Specified value

##### Throws

IllegalArgumentException      Issued when the identification string cannot be identified

- **setSystemProperty**

***public static boolean setSystemProperty(String id, String val)  
throws IllegalArgumentException***

This method sets the specified values for a terminal. An idstring that can be used as a parameter should be modeled after the string used in MH\_sysSetInformation() of HAL document API. It can be added or expanded depending on the communication service

provider or a vendor.

**Parameters**

id	Identification string for a specified value
val	Value to be set

**Return Value**

**true**

When the setting is successful

**false**

When the setting is not successful

**Throws**

IllegalArgumentException	Issued when the identification string cannot be identified
--------------------------	--

**Class BackLight**

java.lang.Object

|

+---org.kwis.msp.handset.**BackLight**public class **BackLight** extends Object

A class used to make adjustments for the LCD backlight

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Method**● **on*****public static void on(int id, int color, int duration) throws IOException***

This is a method that controls the backlight. In every terminal, there is a default backlight setting. This setting maintains its value for a few seconds when a key is pressed. This method should invalidate this value. When going back to the original state after the value is used in an application, the () method should first be called. When time is up, the backlight automatically goes off. For the backlight number, 0 represents the main LCD backlight, whereas 1 represents the auxiliary LCD backlight. When the backlight can specify a particular color, the color is specified by Parameters color. The format for color value is 0xYYRRGGBB (network byte ordering). YY can be ignored. RR, GG, and BB specify red, green, and blue, respectively. Invalid id, color, and duration are ignored.

**Parameters**

id	Backlight number
color	Backlight color
duration	Time when the backlight is on; this time is set in millisecond

**Throws**

IOException Issued in case of an error

● **off*****public static void off() throws IOException***

Turns off the backlight

**Throws**

IOException Issued in case of an error



- **before**

***public static void before() throws IOException***

Maintains the backlight in a state when the program has yet to be executed

**Throws**

IOException Issued in case of an error

- **alwaysOn**

***public static void alwaysOn() throws IOException***

Keeps the backlight on

**Throws**

IOException Issued in case of an error

**Class Call**

```

java.lang.Object
|
+--org.kwis.msp.handset.Call

public class Call extends Object

```

A class that is related to a telephone call

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator**

*None*

**Detailed Description of the Method**

- **accept**

***public static void accept() throws IOException***

Accepts an incoming call

**Throws**

IOException	Issued when this method is called by a program other than the application program, or when an error occurs
-------------	--

- **reject**

***public static void reject() throws IOException***

Rejects an incoming call

**Throws**

IOException	Issued when this method is called by a program other than the application program, or when an error occurs
-------------	--

- **end**

***public static void end() throws IOException***

Ends a call while it is being made, or the conversation is ongoing

**Throws**

IOException	Issued when this method is called by a program other than the application program, or when an error occurs
-------------	--

- **place**

***public static void place(String phonenumber) throws IOException***

Makes a call

**Parameters**

phonenumber	Telephone number
-------------	------------------

**Throws**

IOException	Issued when a call cannot be made
-------------	-----------------------------------

**Class LED**

java.lang.Object

|

+---org.kwis.msp.handset.**LED**public class **LED** extends Object

A class that is used to adjust LED (Light Emitting Diodes)

**Methods inherited from class java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Detailed Description of the Generator***None***Detailed Description of the Method**● **getCount*****public static int getCount() throws IOException***

Returns the number of LEDs in the system

**Throws**

IOException Issued in case of an error

● **set*****public static int set(int leds) throws IOException***

Sets LED to On/Off; when each bit is 1, it represents On (when each bit is 0, it represents Off)

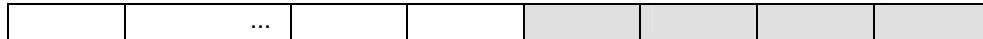
Ex.)

When there are four external LEDs, the system uses four bits beginning with LSB

BIT.

31 bit ...

0



set(0x3) #0 and #1 turn on two LEDs, whereas #2 and #3 turn off two LEDs.

In case of a terminal supporting color, however, the brightest value specified by the terminal company will be set.

**Parameters**

leds LED turned on when the bit is 1; LED turned off when the bit is 0

**Return Value**

Set value

**Throws**

IOException Issued in case of an error

- **get**

***public static int get() throws IOException***

Reads the On/Off status of LED

**Return Value**

Bit OR value of On/Off status of each LED

**Throws**

IOException Issued in case of an error

- **getSupportColor**

***public static int[] getSupportColor(int led) throws IOException***

Gets the color list provided by a relevant LED of a terminal

**Parameters**

led     LED to be controlled

**Return Value**

Color list provided

**Throws**

IOException Issued in case of an error

- **getColor**

***public static int getColor(int led) throws IOException***

Gets the RGB value specified for a relevant LED

**Parameters**

led     LED to be controlled

**Return Value**

RGB value

**Throws**

IOException Issued in case of an error

- **setColor**

***public static int setColor(int led, int rgb) throws IOException***

Sets the RGB value for a relevant LED in a terminal; in the absence of an RGB value received as an argument, the nearest value will be set

**Parameters**

led     LED to be controlled

rgb     RGB value

**Return Value**

Set RGB value

**Throws**

IOException   Issued in case of an error

## **3.2. MIDP**

MIDP Specifications shall provide identical methods in compliance with Specification 2.0 of Sun Microsystems.

Reference website: <http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html>

## 4.1. Extended Unicode Supporting Korean (EUC\_KR)

```
Unicode Character Value = KSC5601 Character Value | 0xE000 SC5601
Character Value = Unicode Character Value & 0xFFFF
```

Table 4-1-1. Character code table

447



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
--	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

ABA0	ア	アイ	ウ	エ	エ	オ	カ	キ	ク
ABB0	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ
ABC0	ス	ズ	セ	ゼ	ソ	タ			
ABD0	ダ	チ	ツ	ツ	テ	デ	ト	ナ	ニ
ABE0	ヌ	ネ	ノ	ハ					
ABF0	バ	パ	ヒ	ビ	フ	ブ	ヘ	ベ	ホ
	ボ	ポ	マ	ミ					
	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル
	レ	ロ	ワ						
	ヰ	ヱ	ラン	ヴ	カ	ケ	?		

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
ACA0	А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	
ACB0	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э
ACC0	Ю	Я	?	?	?	?	?	?	?	?	?	?	?	?	?	?
ACD0	?	а	б	в	г	д	е	ё	ж	з	и	й	к	л	м	н
ACE0	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э
ACF0	ю	я														